WESTERN SYDNEY
UNIVERSITY

Module 3

Creating Arrays

Variables as Arrays

In MATLAB, a variable is stored as an array of numbers. When appropriate, it is interpreted as a scalar, vector or matrix.

scalar	vector	matrix
1×1	$n \times 1$ or $1 \times n$	$n \times m$

The size of an array is specified by the number of rows and the number of columns in the array, with the number of rows indicated first.

Defining (or assigning) arrays

- An array can be defined by typing in a list of numbers enclosed in square brackets:
  - Commas or spaces separate numbers.

```
A = [12, 18, -3]   or   A = [12 18 -3]
A =
    12    18    -3
```

  - Semicolons indicate a new row.

```
B = [2, 5, 2; 1 , 1, 2; 0, -2, 6]
B =
     2     5     2
     1     1     2
     0    -2     6
```

Scalars

- Scalars are 1×1 arrays.
- They contain a single value, for example:

```
r = 6
height = 5.3
width = 9.07
```

Scalars

- Important scalars to know:

Symbol or name	**MATLAB command**
π	pi
Euler's number *e*	exp(1)
infinity	inf
imaginary numbers	i or j

Vectors

- A vector is a list of numbers expressed as a 1 dimensional array.
- A vector can be n×1 or 1×n.
- Columns are separated by commas (or spaces):

```
h = [1, 2, 3]
```

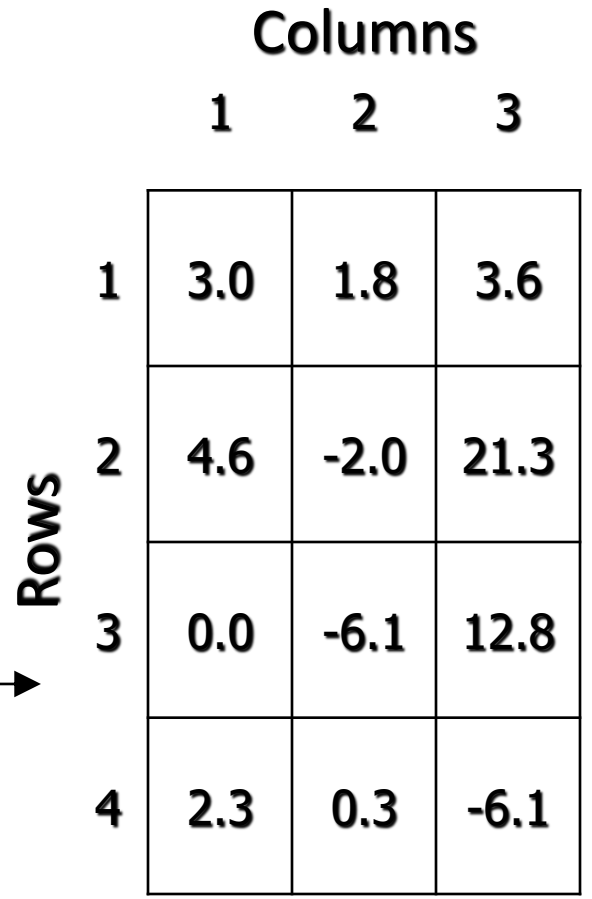
- Rows are separated by semicolons:

```
v = [1; 2; 3]
```

Matrices

- A matrix is a two dimensional array of numbers.

- For example, this is a 4×3 matrix:




```
m = [3.0, 1.8, 3.6; 4.6, -2.0, 21.3; 0.0,
-6.1, 12.8; 2.3, 0.3, -6.1]
```

Exercise

- Enter the following into MATLAB:
  - Scalar:

```
a = 1
```

  - Vectors:

```
b = [1, 0, 2]
c = [1 0 2]
```

  - Matrix:

```
d = [5, 4, 3; 0, 2, 8]
```

Defining arrays

- You can define an array in terms of other arrays:

```
C = [A; B]
C =
    12    18    -3
     2     5     2
     1     1     2
     0    -2     6

D = [C, C]
D =
    12    18    -3    12    18    -3
     2     5     2     2     5     2
     1     1     2     1     1     2
     0    -2     6     0    -2     6
```

Colon (:) operator

- Colon notation can be used to define evenly spaced vectors in the form:

first : last

```
H = 1:6
H =
     1     2     3     4     5     6
```

- The default spacing is 1. To use a different increment use the form:

first : increment : last

```
I = 1:2:11
I =
     1     3     5     7     9    11
```

  - The numbers now increment by 2

Transpose operator

- The transpose operator, an apostrophe, changes all of an array's rows to columns and columns to rows.

```
J = [1 , 3, 7]
J =
     1     3     7

J'
ans =
     1
     3
     7
```

Manipulating Arrays

- The functions fliplr() and flipud() flip a matrix left-to-right and top-to-bottom, respectively.
  - Experiment with these functions to see how they work.

Useful functions

- linspace(a,b,c) – generates a linearly spaced vector from *a* to *b* with *c* numbers. c = 100 if c is not specified
- logspace(a,b,c) – generates logarithmically spaced vector from 10^a to 10^b of c numbers. c = 50 is c is not specified
- magic(n) – creates an n×n arrayof numbers from 1 to n^2
- randi([a b],n,m) – creates an n×m array of random integers between and including a and b
- rand(n,m) – creates an n×m array of random integers between 0 and 1
- zeros(n,m) – creates an n×m array of zeros
- ones(n,m) – creates an n×m array of ones

Creating arrays of zeros & ones

- Create an array of zeros:

```
E = zeros(3,5)
E =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

- Create an array of ones:

```
F = ones(2,3)
F =
     1     1     1
     1     1     1
```

Note: Placing a single number inside either function will return an $n \times n$ array. e.g. **`ones(4)`** will return a 4×4 array filled with ones.