Variables and Arrays

Variables are named locations in memory where numbers, strings and other elements of data may be stored while the program is working.

Variable names are combinations of letters and digits, but must start with a letter.

Variable names must be unique in the first 31 characters

MATLAB does not require you to declare the names of variables in advance of their use.

To assign a variable a value, use the assignment statement. This takes the form

variable=expression;

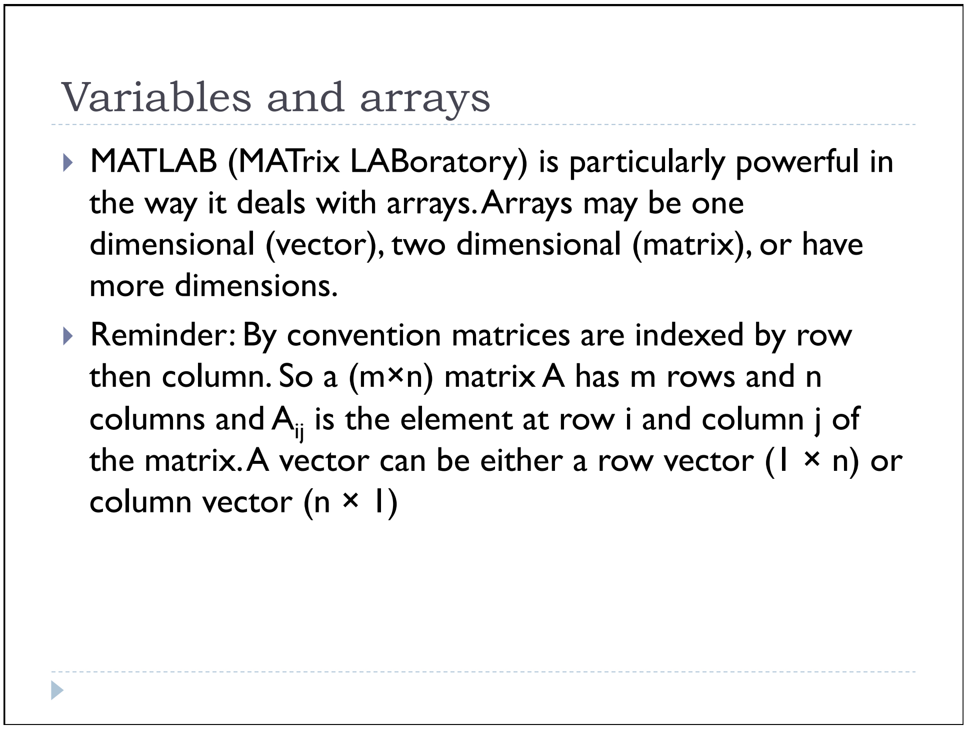
Variables and Arrays

For example:

- a=6;
- name='Mark';

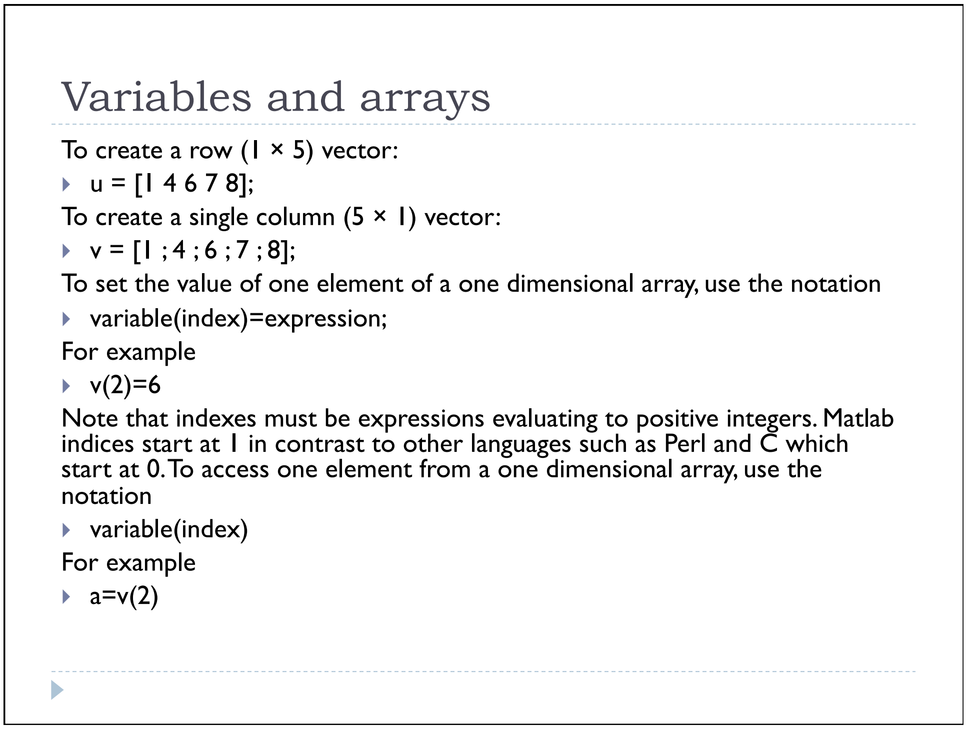
To display the result of an assignment, you omit the semicolon at the end of the line. You can also use this feature to display variables. Another option is to use the disp() function

- b=6
- name2='Tom'
- name
- disp(name)



Variables and arrays

- MATLAB (MATrix LABoratory) is particularly powerful in the way it deals with arrays. Arrays may be one dimensional (vector), two dimensional (matrix), or have more dimensions.
- Reminder: By convention matrices are indexed by row then column. So a (m×n) matrix A has m rows and n columns and A_{ij} is the element at row i and column j of the matrix. A vector can be either a row vector (1 × n) or column vector (n × 1)



Variables and arrays

To create a row (1 × 5) vector:

- u = [1 4 6 7 8];

To create a single column (5 × 1) vector:

- v = [1 ; 4 ; 6 ; 7 ; 8];

To set the value of one element of a one dimensional array, use the notation

- variable(index)=expression;

For example

- v(2)=6

Note that indexes must be expressions evaluating to positive integers. Matlab indices start at 1 in contrast to other languages such as Perl and C which start at 0. To access one element from a one dimensional array, use the notation

- variable(index)

For example

- a=v(2)

Variables and arrays

To create a 2x3 matrix:

- m = [1 3 4; 5 7 9]

You can check the dimensions of a vector or matrix object by using the size function, which, for vectors and 2D matrices returns a 1 × 2 matrix with the dimensions (m × n). Try:

- size(u)
- size(v)
- size(m)

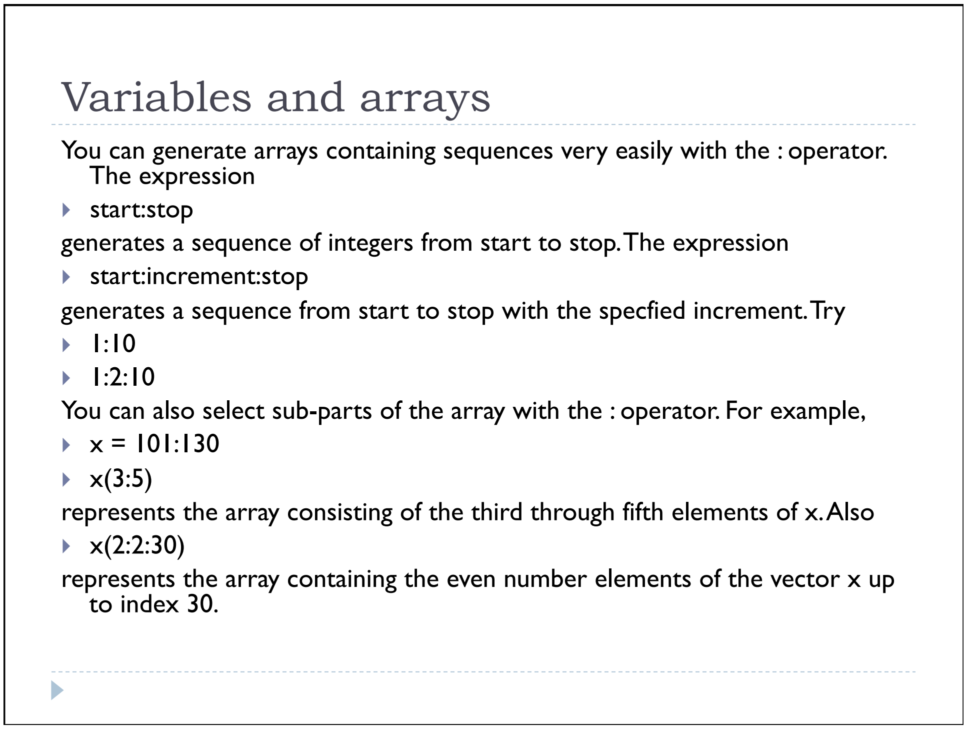
Variables and arrays

The expressions used to initialize arrays can include algebraic operations and all or portions of previously defined arrays. For example:

- a = [0 1+7];
- b = [a(2) 7 a];
- d = [b ; b];

Note horizontal and vertical concatenation using space or semi-colon

- Not all elements in an array must be defined when it is created, for example, if c is not previously defined:
- c(2,3) = 5;
- produces a 2x3 matrix, c= $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 5 \end{bmatrix}$



Variables and arrays

You can generate arrays containing sequences very easily with the : operator. The expression

- start:stop

generates a sequence of integers from start to stop. The expression

- start:increment:stop

generates a sequence from start to stop with the specfied increment. Try

- 1:10
- 1:2:10

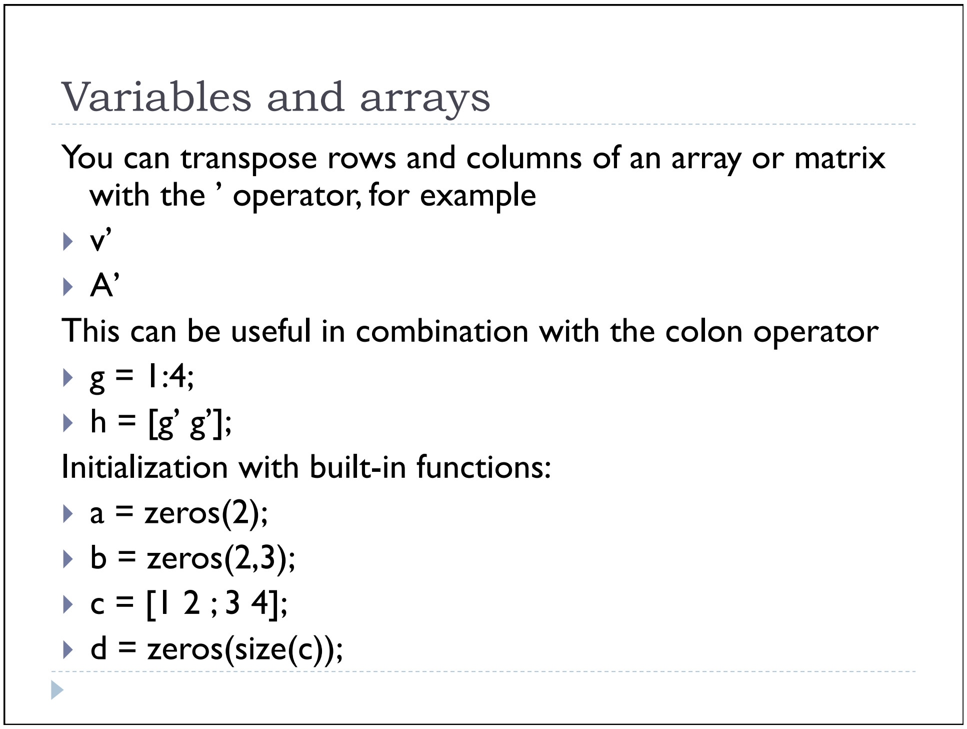
You can also select sub-parts of the array with the : operator. For example,

- x = 101:130
- x(3:5)

represents the array consisting of the third through fifth elements of x. Also

- x(2:2:30)

represents the array containing the even number elements of the vector x up to index 30.



Variables and arrays

You can transpose rows and columns of an array or matrix with the ’ operator, for example

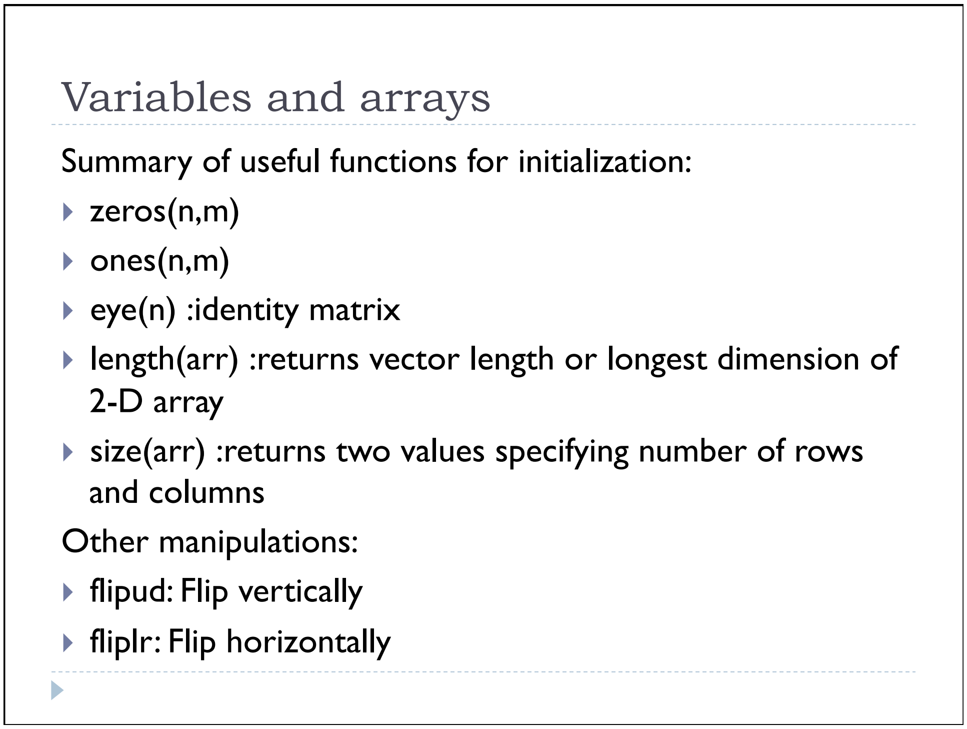
- v’
- A’

This can be useful in combination with the colon operator

- g = 1:4;
- h = [g’ g’];

Initialization with built-in functions:

- a = zeros(2);
- b = zeros(2,3);
- c = [1 2 ; 3 4];
- d = zeros(size(c));



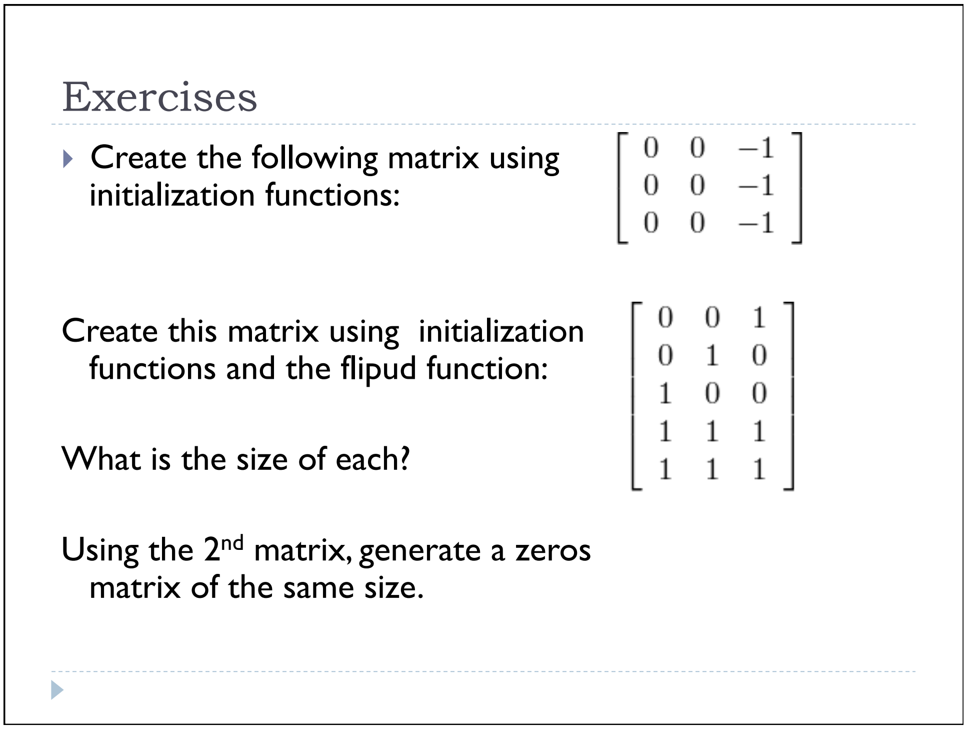
Variables and arrays

Summary of useful functions for initialization:

- zeros(n,m)
- ones(n,m)
- eye(n) :identity matrix
- length(arr) :returns vector length or longest dimension of 2-D array
- size(arr) :returns two values specifying number of rows and columns

Other manipulations:

- flipud: Flip vertically
- fliplr: Flip horizontally



Exercises

- Create the following matrix using initialization functions:

$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & -1 \\ 0 & 0 & -1 \end{bmatrix}$$

Create this matrix using initialization functions and the flipud function:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

What is the size of each?

Using the 2nd matrix, generate a zeros matrix of the same size.

Multidimensional arrays

So far we've created row/column vectors and 2-D arrays

We can create arrays with as many dimensions as necessary, for example:

- c(:,:,1) = [1:3 ; 4:6];
- c(:,:,2) = [7:9 ; 10:12];
- whos c
- c

creates a 2x3x2 matrix

Storage of arrays in memory

- Arrays are always stored linearly internally. We can access the contents of any array with a single subscript.
- The elements of the array are stored in column major order, i.e. the order is column 1, column 2, etc. (as opposed to row1, row2, etc.)
- For example:
- a = [1:3 ; 4:6]; creates the matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- Evaluating a(4) gives "5" not "4"

Subarrays and indexing

We can select and use subsets of arrays as if they are separate arrays. To do this, we include a list of all the elements to be selected in parentheses after the array name, e.g. if we create

- arr1 = [1.1 -2.2 3.3 -4.4 5.5]

Then arr1([1 4]) is the array [1.1 -4.4] and

- arr1[1:2:5] is the array [1.1 3.3 5.5]

For 2-D matrices, e.g, if we create

- arr2 = magic(4)

then we can select the 2nd column with arr2(:,2)

Here the colon by itself means select the entire dimension (in this case all rows = column)

Similarly, to select only the 1st and 3rd rows: arr2([1 3],:)

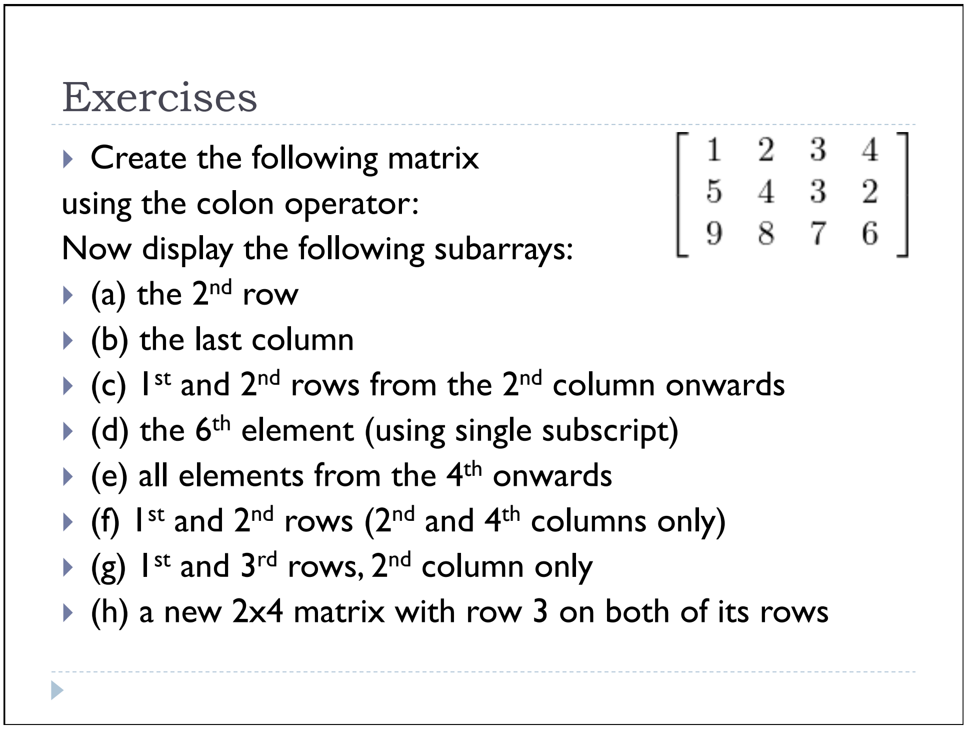
Subarrays and indexing

Subarrays can be used on the LHS of an assignment, e.g.

- arr2(2:3,2:3) = [1 2 ; 3 4];
- arr2([1 3],1:2) = [1 2 ; 3 4];
- Generally speaking the LHS and RHS dimensions must match. The exception is that one can assign a scalar to an entire subarray, e.g.
- arr2(2:3,2:3) = 1;

The *end* function returns the highest value taken by that subscript, e.g. if m is declared as magic(3); then

- m(2:end,:); returns all rows except the 1st , and
- m(:,end) returns the last column



Exercises

- Create the following matrix

using the colon operator:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 4 & 3 & 2 \\ 9 & 8 & 7 & 6 \end{bmatrix}$$

Now display the following subarrays:

- (a) the 2nd row
- (b) the last column
- (c) 1st and 2nd rows from the 2nd column onwards
- (d) the 6th element (using single subscript)
- (e) all elements from the 4th onwards
- (f) 1st and 2nd rows (2nd and 4th columns only)
- (g) 1st and 3rd rows, 2nd column only
- (h) a new 2x4 matrix with row 3 on both of its rows