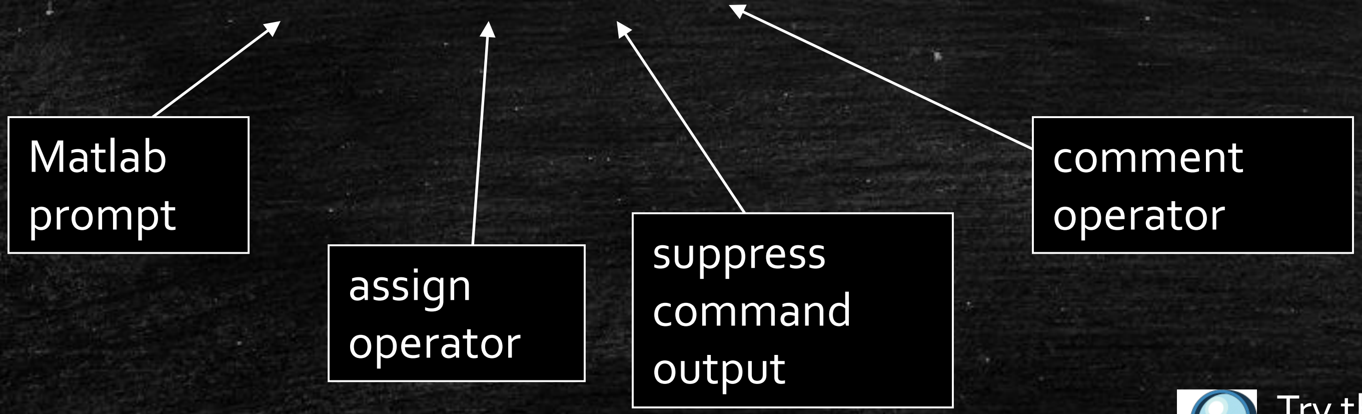
Introduction to Variables & Matrices

Variables

- Don’t have to declare type
- Don’t even have to initialise
- Just assign in command window

```
>>
>> a=12; % variable a is assigned 12
```



Matlab prompt

assign operator

suppress command output

comment operator

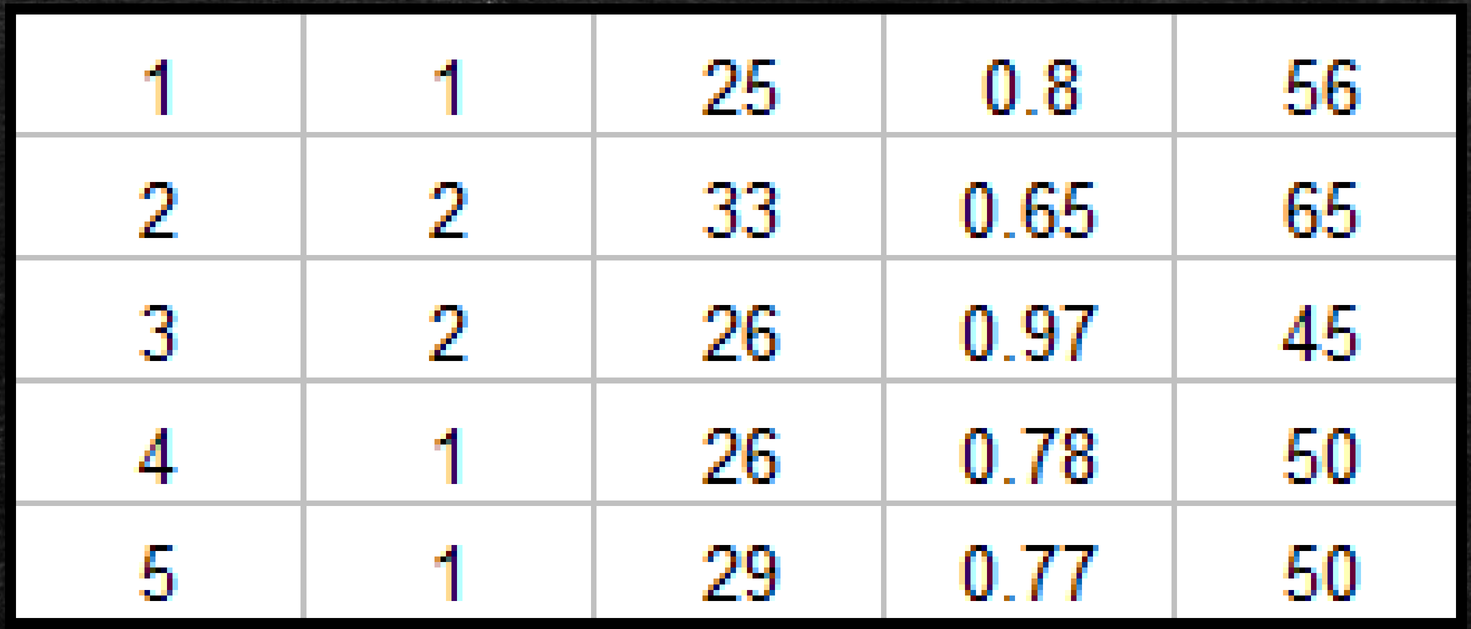
Try the same line without the semicolon and comments

Variable Complexity

According to the terminology used in linear algebra, some special cases of matrices in terms of their number of dimensions are referred to as:

- **Scalar**: «null-dimensional» matrix, i.e. only one element
- **Vector**: one-dimensional matrix (array)
- **Matrix**: matrix with two or more dimensions. Sometimes, they are also called **2D or 3D arrays**.

Example:
A simple matrix with data from five subjects



1	1	25	0.8	56
2	2	33	0.65	65
3	2	26	0.97	45
4	1	26	0.78	50
5	1	29	0.77	50

Variables (continued …)

- View variable contents by simply typing the variable name at the command prompt

```
>> a
a =
12
>>
>> a*2
a =
24
>>
```

Creating a Matrix

- Don’t need to initialise type, or dimensions

- Simply type:

```
>>A = [3 2 1; 5 1 0; 2 1 7]
```

square brackets to define matrices

semicolon for next row in matrix

Octave will respond with a matrix in pretty-print:

```
A =

   3   2   1

   5   1   0

   2   1   7

>>
```

Variables and Data Types

Creating a Character String

- Simply type:

```
octave:4> str = 'Hello World'
```

Opposed to Matlab, Octave can also deal with double quotes. For compatibility reasons, **use single quotes**.

Creating a Structure

- Type for instance:

```
octave:5> data.id = 3;
octave:6> data.timestamp = 1265.5983;
octave:7> data.name = 'sensor 1 front';
```

Variables and Data Types

Display Variables

- Simply type its name:

```
octave:1> a
a = 4
```

Suppress Output

- Add a semicolon:

```
octave:2> a;
octave:3> sin(phi);
```

Applies also to function calls.

Variables and Data Types

- **Variables** have **no permanent type.**

  `s = 3` followed by `s = 'octave'` is fine

- Use `who` (or the more detailed `whos`) to **list** the **currently defined variables**. Example output:

```
Variables in the current scope:

 Attr  Name          Size                    Bytes  Class
 ====  ====          ====                    =====  =====
       A             3x3                       72   double
       a             1x1                        8   double
       ans          21x1                      168   double
       s             1x5                        5   char
       v             1x21                      24   double
```

Manipulating Matrices

```
A =

  3  2  1
  5  1  0
  2  1  7
```

- Access elements of a matrix

```
>>A(1,2)

ans=

2
```

indices of matrix element(s)

- Remember Matrix(row,column)
- Naming convention Matrix variables start with a capital letter while vectors or scalar variables start with a simple letter

The : operator

- VERY important operator in Matlab
- Means ‘to’

```
>> 1:10

ans =

   1   2   3   4   5   6   7   8   9   10

>> 1:2:10

ans =

   1   3   5   7   9
```

Try the following

```
>> x=0:pi/12:2*pi;
>> y=sin(x)
```

The : operator and matrices

```
>>A(3,2:3)

ans =

  1   7

>>A(:,2)

ans =

  2

  1

  1
```

```
A =

  3   2   1

  5   1   0

  2   1   7
```

What'll happen if you type A(:,:) ?

Matrices

Indexing

Always "row before column"!

`aij = A(i,j)`	Get an element
`r = A(i,:)`	Get a row
`c = A(:,j)`	Get a column
`B = A(i:k,j:l)`	Get a submatrix

- **Useful indexing command** `end` **:**

```
octave:1> data=[4-1 35 9     11 -2];
octave:2> v = data(3:end)
v =
     35   9   11   -2
```

Manipulating Matrices

```
A =

   3   2   1
   5   1   0
   2   1   7


B =

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

```
>> A '              % transpose
>> B*A % matrix multiplication
>> B.*A% element by element multiplication
>> B/A  % matrix division
>> B./A % element by element division
>> [B A]% Join matrices (horizontally)
>> [B; A]           % Join matrices (vertically)
```

🔍 Enter matrix B into the Matlab workspace

🔍 Create matrices A and B and try out the the matrix operators in this slide

Matrices

Assigning a Row/Column

- All referenced elements are set to the scalar value.

```
octave:1> A = [1 2 3 4 5; 2 2 2 2 2; 3 3 3 3 3];
octave:2> A(3,:) = -3;
```

Adding a Row/Column

- If the referenced row/colum doesn't exist, it's added.

```
octave:3> A(4,:) = 4
A =
    1   2   3   4   5
    2   2   2   2   2
   -3  -3  -3  -3  -3
    4   4   4   4   4
```

Matrices

Deleting a Row/Column

- Assigning an empty matrix `[]` deletes the referenced rows or columns. Examples:

```
octave:4> A(2,:) = []

A =

    1   2   3   4   5
   -3  -3  -3  -3  -3
    4   4   4   4   4

octave:4> A(:,1:2:5) = []

A =
   2   4
   2   2
  -3  -3
   4   4
```

Matrices

Get Size

`nr = size(A,1)`	Get number of rows of A
`nc = size(A,2)`	Get number of columns of A
`[nr nc] = size(A)`	Get both (remember order)
`l = length(A)`	Get whatever is bigger
`numel(A)`	Get number of elements in A
`isempty(A)`	Check if A is empty matrix []

Octave only:

`nr = rows(A)`	Get number of rows of A
`nc = columns(A)`	Get number of columns of A

Matrices

Matrix Operations

`B = 3*A`	Multiply by Scalar
`C = A+B A*B`	Add and multiply
`B = A'`	Transpose A
`B = inv(A)`	Invert A
`s = v'*Q*v`	Mix vectors and matrices
`d = det(A)`	Determinant of A
`[v lambda] = eig(A)`	Eigenvalue decomposition
`[U S V] = svd(A)`	Sing. value decomposition

- many many more...

Matrices

Vector Operations

With `x` being a column vector

- `s = x'*x` Inner product, result is a scalar
- `X = x*x` Outer product, result is a matrix
- `e = '` Gives an error

  `x*x`

Element-Wise Operations (for vectors/matrices)

- `s = x.+x` Element-wise addition
- `p = x.*` Element-wise multiplication
- `q = x` Element-wise division

  `x./`

  `x`
- `e = x.^3` Element-wise power operator

Matrices

Useful Vector Functions

`min(v)`	Return smallest element in v
`max(v)`	Return largest element in v
`sort(v,'ascend')`	Sort in ascending order
`sort(v,'descend')`	Sort in descending order
`find(v)`	Return vector of indices of all non-zero elements in v. Great in combination with **vectorized conditions**. Example: `ivec = find(datavec == 5).`

Matrices

Special Matrices

- `A = zeros(m,n)` Zero matrix of size m x n
- `B = ones(m,n)` Matrix of size m x n with all 1's
- `I = eye(n)` Identity matrix of size n
- `D = diag([a b c])` Diagonal matrix of size 3 x 3 with a,b,c in the main diagonal

Just for fun

- `M = magic(n)` Magic square matrix of size n x n. (All rows and columns sum up to the same number)

Matrices

Random Matrices and Vectors

- `R = rand(m,n)` Matrix with m x n uniformly distributed random numbers from interval [0..1]
- `N = randn(m,n)` Row vector with m x n normally distributed random numbers with zero mean, unit variance
- `v = randperm(n)` Row vector with a random permutation of the numbers 1 to n

Matrices

Multi-Dimensional Matrices

Matrices can have more than two dimensions.

- **Create a 3-dimensional matrix** by typing, e.g.,

```
octave:1> A = ones(2,5,2)
```

Octave will respond by

```
A =
ans(:,:,1) =
    1    1    1    1    1
    1    1    1    1    1

ans(:,:,2) =
    1    1    1    1    1
    1    1    1    1    1
```

Matrices

Multi-Dimensional Matrices

- All operations to **create, index, add, assign, delete and get size** apply **in the same fashion**

Examples:

- `[m n l] = size(A)`
- `A = rand(m,n,l)`
- `m = min(min(min(A)))`
- `aijk = A(i,j,k)`
- `A(:,:,5) = -3`

Matrices

Rearranging Matrices

- `reshape(A,m,n)` **Change size** of matrix A to have dimension m x n. An error results if A does not have m x n elements
- `circshift(A,[m n])` **Shift elements** of A m times in row dimension and n times in column dimension
- `shiftdim(A,n)` Shift the dimension of A by n. **Generalizes transpose** for multi-dimensional matrices

Matrices

Rearranging Matrices Example

Let P = [x1; y1; x2; y2; ...] be a 2nx1 column vector of n (x,y)-pairs. Make it a column vector of (x,y,theta)-tuples with all theta values being pi/2:

- Make it a 2xn matrix
  ```
  octave:1> P = reshape(P,2,numel(P)/2);
  ```
- Add a third row, assign pi/2
  ```
  octave:2> P(3,:) = pi/2;
  ```
- Reshape it to be a 3nx1 column vector
  ```
  octave:3> P = reshape(P,numel(P),1);
  ```