# APPLICATIONS OF MATLAB IN ENGINEERING

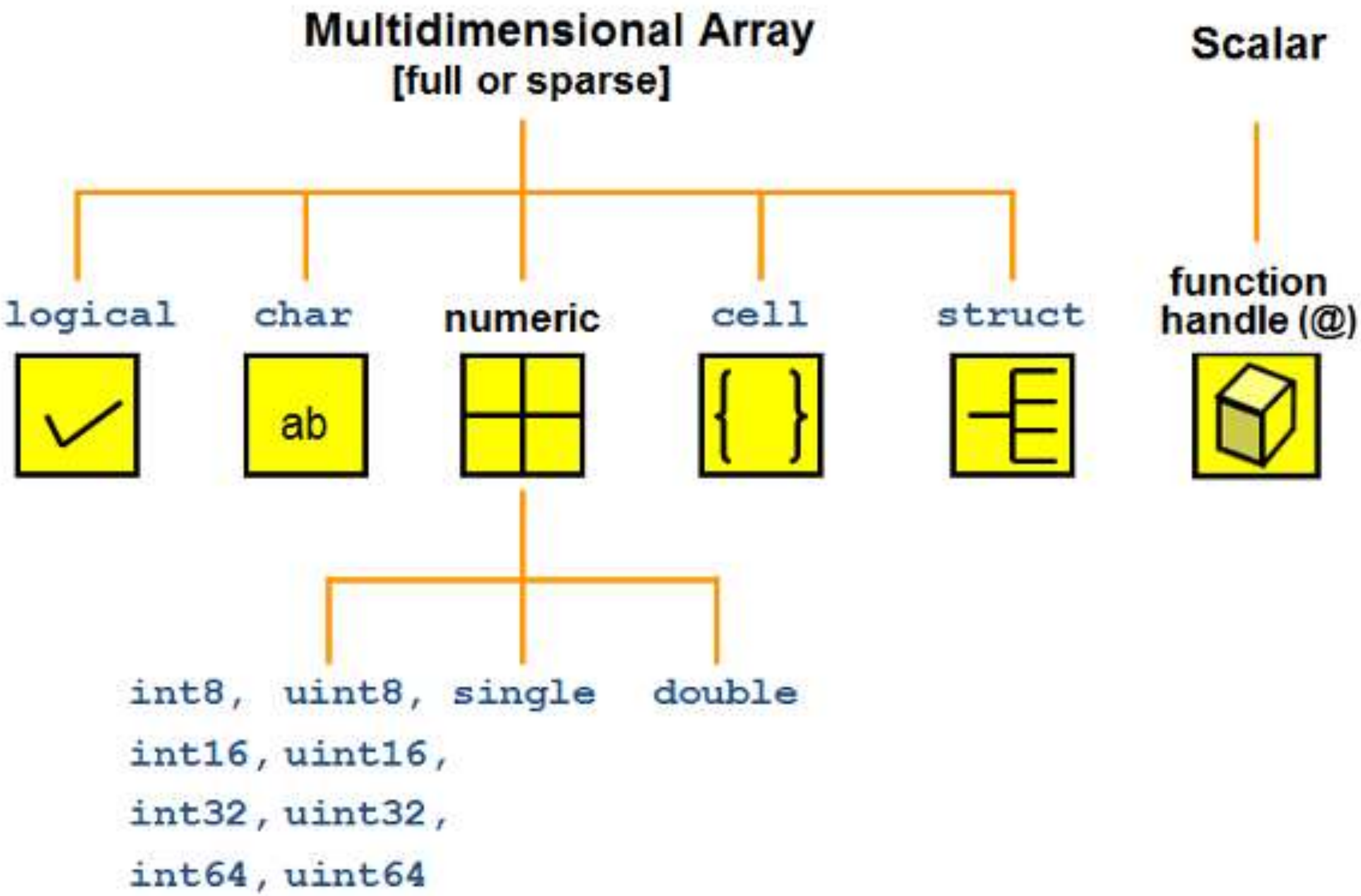
Yan-Fu Kuo
Dept. of Bio-industrial Mechatronics Engineering
National Taiwan University

Fall 2015

Today:

- Variables: string, structure, cell
- Data access

# MATLAB Data (Variables) Types



# Variable (Data) Type Conversion

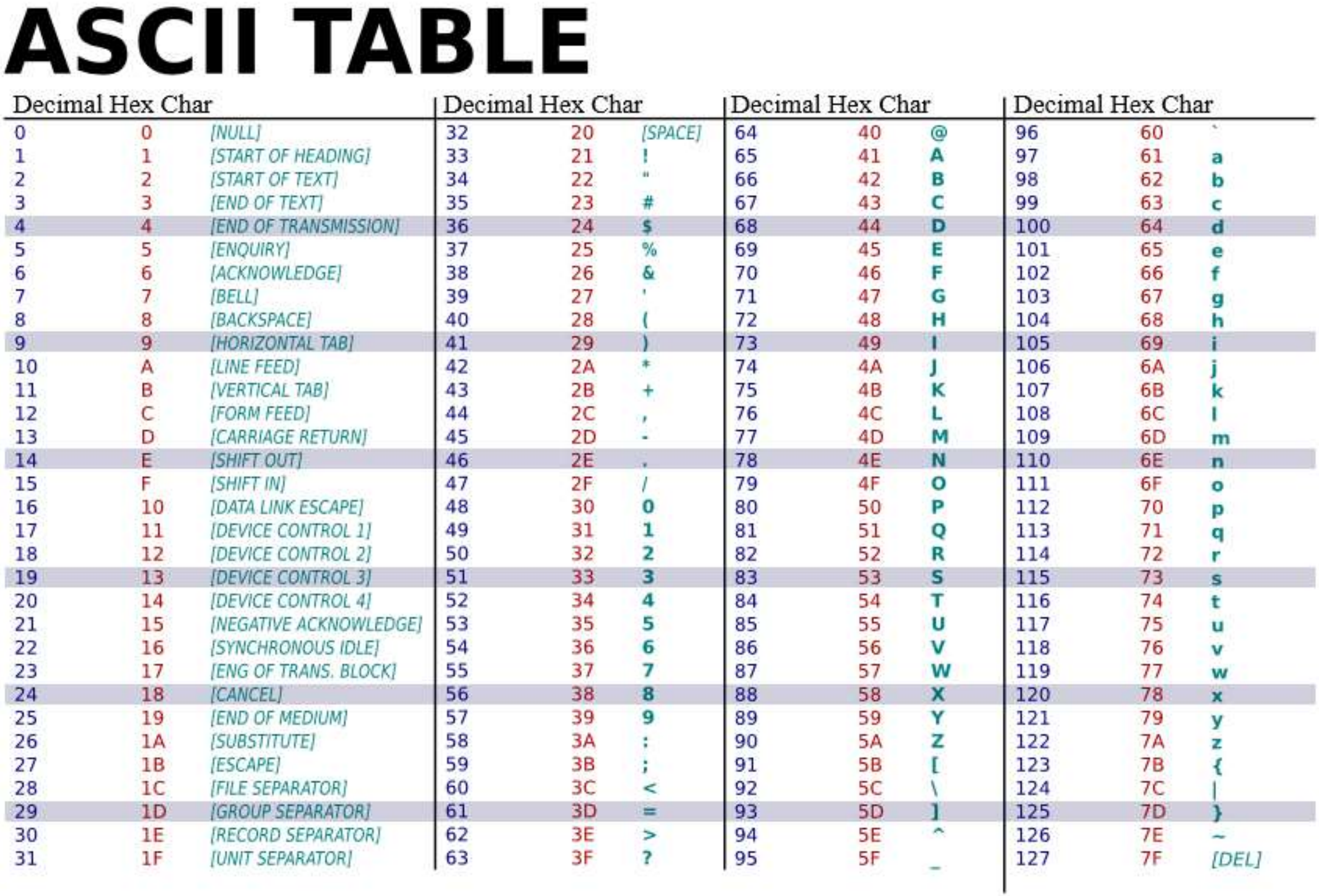
| | |
|---|---|
| double() | Convert to double precision |
| single() | Convert to single precision |
| int8() | Convert to 8-bit signed integer |
| int16() | Convert to 16-bit signed integer |
| int32() | Convert to 32-bit signed integer |
| int64() | Convert to 64-bit signed integer |
| uint8() | Convert to 8-bit unsigned integer |
| uint16() | Convert to 16-bit unsigned integer |
| uint32() | Convert to 32-bit unsigned integer |
| uint64() | Convert to 64-bit unsigned integer |

# Character (char)

- A character is represented in ASCII using a numeric code between 0 to 255
- Create a character or a string by putting them into a pair of apostrophe:

```
s1 = 'h'
whos
uint16(s1)
```

```
s2 = 'H'
whos
uint16(s2)
```



# ASCII TABLE

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [ENG OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |

# String

- An array collects characters:

```
s1 = 'Example';
s2 = 'String';
```

- String concatenation:

```
s3 = [s1 s2];
```

```
s4 = [s1; s2];
```

# Logical Operations and Assignments

- Many numerical and logical operators can be applied to strings

```
str = 'aardvark';
'a' == str
```

- Try this:

```
str(str == 'a') = 'Z'
```

- What if we want to compare the entire string with another?

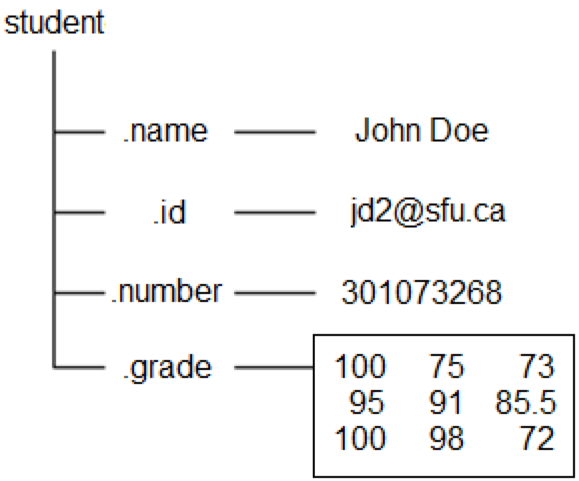
# Exercise

- Write a script that inverts any given string

```
s1='I like the letter E'
          ⇓
s2='E rettel eht ekil I'
```

# Structure

- A method of storing heterogeneous data
- Structures contain arrays called fields
- Student assignment grades:



student

.name — John Doe

.id — jd2@sfu.ca

.number — 301073268

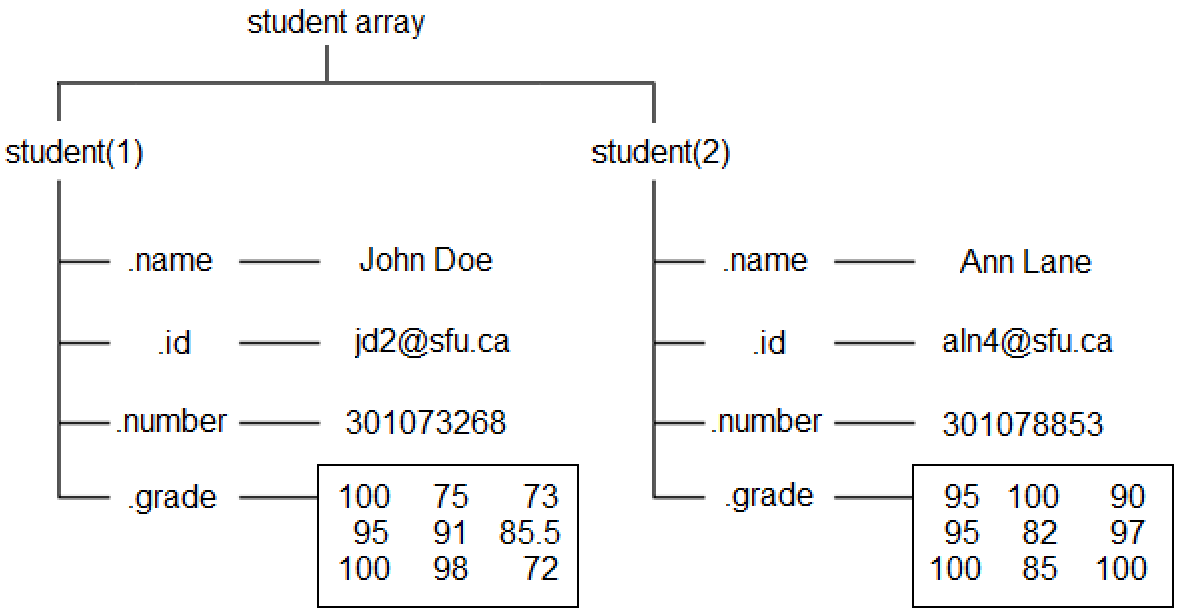
.grade —

| | | |
|---|---|---|
| 100 | 75 | 73 |
| 95 | 91 | 85.5 |
| 100 | 98 | 72 |

```
student.name = 'John Doe';
student.id = 'jdo2@sfu.ca';
student.number = 301073268;
student.grade = [100, 75, 73; ...
                 95, 91, 85.5; ...
                 100, 98, 72];

student
```

# Adding Information to A Structure



student array

student(1)

.name — John Doe

.id — jd2@sfu.ca

.number — 301073268

.grade —

| | | |
|---|---|---|
| 100 | 75 | 73 |
| 95 | 91 | 85.5 |
| 100 | 98 | 72 |

student(2)

.name — Ann Lane

.id — aln4@sfu.ca

.number — 301078853

.grade —

| | | |
|---|---|---|
| 95 | 100 | 90 |
| 95 | 82 | 97 |
| 100 | 85 | 100 |

```
student(2).name = 'Ann Lane';
student(2).id = 'aln4@sfu.ca';
student(2).number = 301078853;
student(2).grade = [95 100 90; 95 82 97; 100 85 100];
```

- Retrieve the 3rd grade for Ann Lane

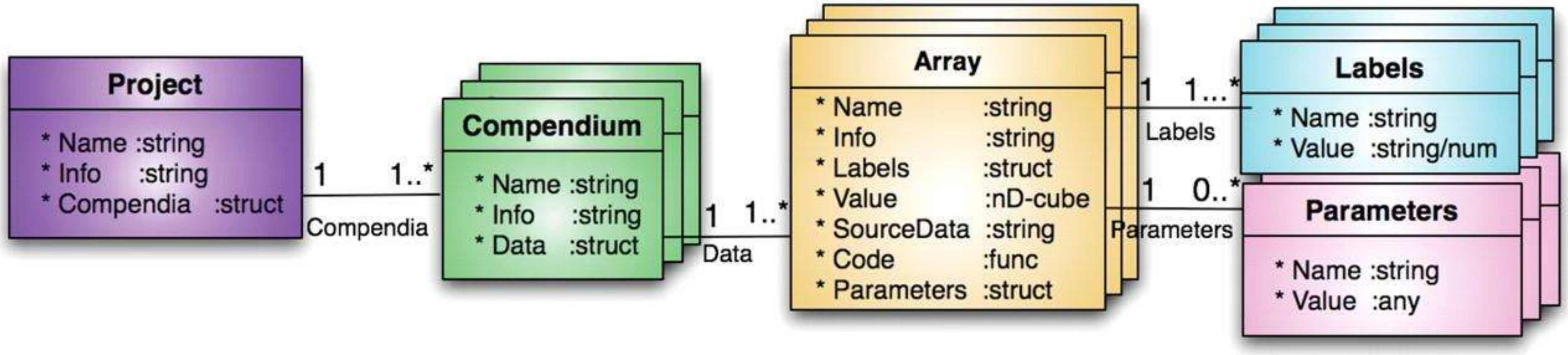
# Structure Functions

| | |
|---|---|
| cell2struct | Convert cell array to structure array |
| fieldnames | Field names of structure, or public fields of object |
| getfield | Field of structure array |
| isfield | Determine whether input is structure array field |
| isstruct | Determine whether input is structure array |
| orderfields | Order fields of structure array |
| rmfield | Remove fields from structure |
| setfield | Assign values to structure array field |
| struct | Create structure array |
| struct2cell | Convert structure to cell array |
| structfun | Apply function to each field of scalar structure |

- Try:

```
fieldnames(student)
rmfield(student,'id')
```

# Nesting Structures



```
A = struct('data', [3 4 7; 8 0 1], 'nest', ...
    struct('testnum', 'Test 1', ...
    'xdata', [4 2 8],'ydata', [7 1 6]));
A(2).data = [9 3 2; 7 6 5];
A(2).nest.testnum = 'Test 2';
A(2).nest.xdata = [3 4 2];
A(2).nest.ydata = [5 0 9];
A.nest
```

# Cell Array

- Another method of storing heterogeneous data
- Similar to matrix but each entry contains different type of data
- Declared using { }

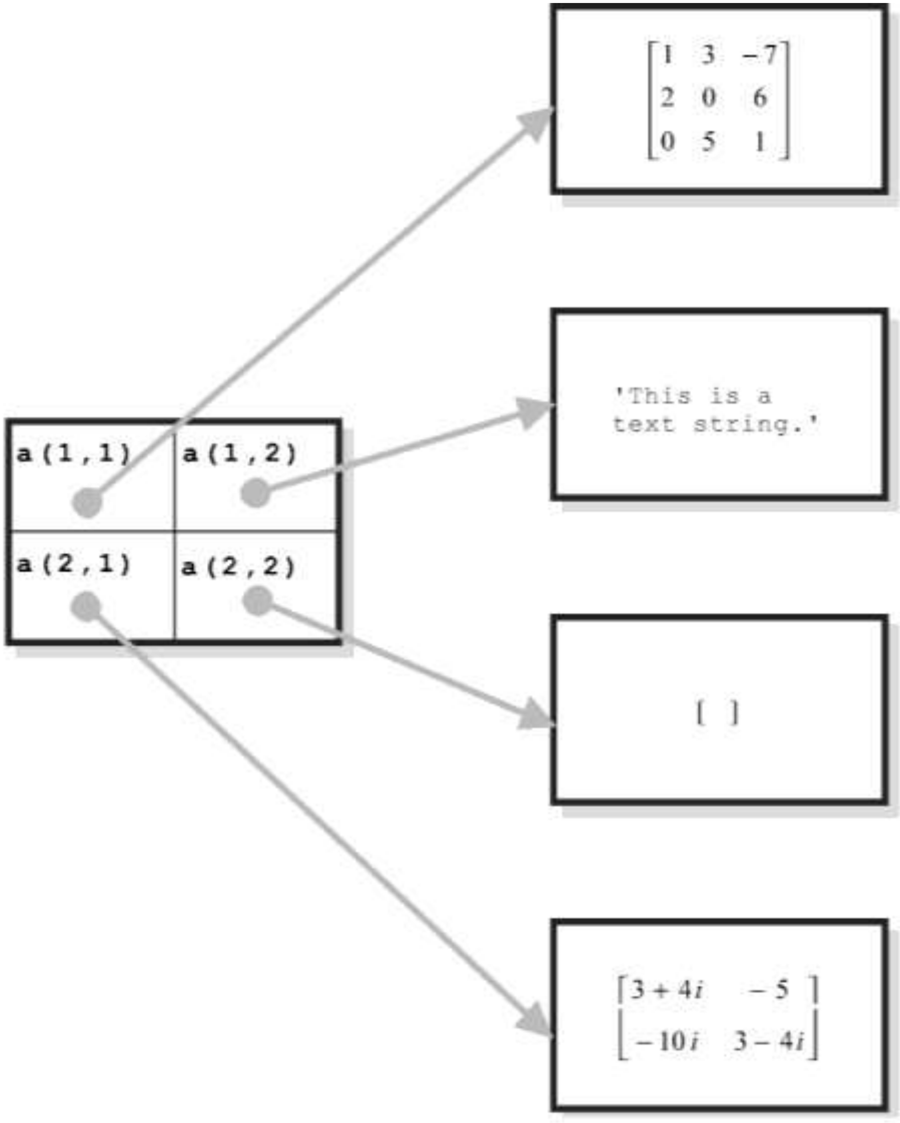
| | |
|---|---|
| $\begin{bmatrix} 1 & 4 & 3 \\ 0 & 5 & 8 \\ 7 & 2 & 9 \end{bmatrix}$ | 'Anne Smith' |
| 3+7i | $[-\pi \quad 0 \quad \pi]$ |

```
A(1,1)={[1 4 3; 0 5 8; 7 2 9]};
A(1,2)={'Anne Smith'};
A(2,1)={3+7i};
A(2,2)={-pi:pi:pi};
A
```

```
A{1,1}=[1 4 3; 0 5 8; 7 2 9];
A{1,2}='Anne Smith';
A{2,1}=3+7i;
A{2,2}=-pi:pi:pi;
A
```

# How Does MATLAB Do It?

- Each entry in a cell array holds a pointer to a data structure
- Different cells of the same cell array can point to different types of data structures


a(1,1)
a(1,2)
a(2,1)
a(2,2)
$\begin{bmatrix} 1 & 3 & -7 \\ 2 & 0 & 6 \\ 0 & 5 & 1 \end{bmatrix}$
'This is a text string.'
[ ]
$\begin{bmatrix} 3+4i & -5 \\ -10i & 3-4i \end{bmatrix}$

# Exercise

- Create a cell array B that has the following structure

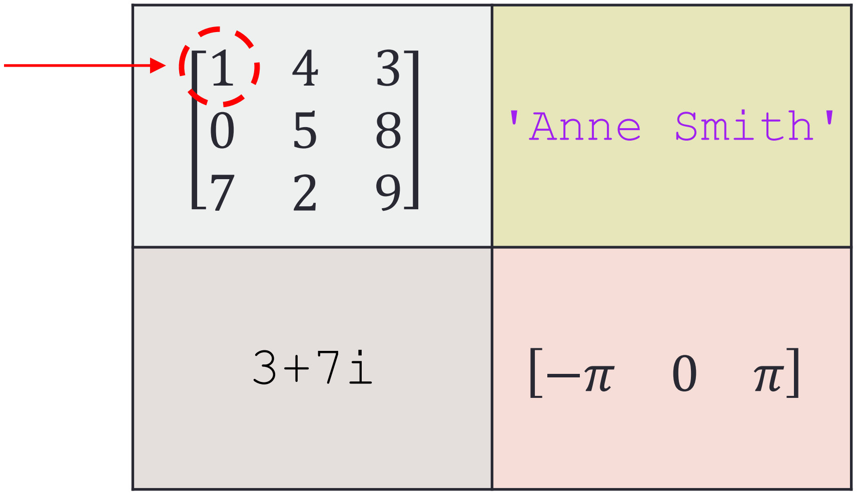
| | |
|---|---|
| 'This is the first cell'<br>(String) | [5+j*6   4+j*5]<br>(1x2 complex number array) |
| 1   2   3<br>4   5   6<br>7   8   9<br>(3x3 integer array) | {'Tim',  'Chris'}<br>(1X2 string array) |

# Accessing Cell Array

- Curly braces, { }, are used to access the "content" of cell arrays
- What are the differences between C and D?

```
C = A{1,1}
D = A(1,1)
```

- How do I get this number?



| | |
|---|---|
| $\begin{bmatrix} 1 & 4 & 3 \\ 0 & 5 & 8 \\ 7 & 2 & 9 \end{bmatrix}$ | 'Anne Smith' |
| 3+7i | $[-\pi \quad 0 \quad \pi]$ |

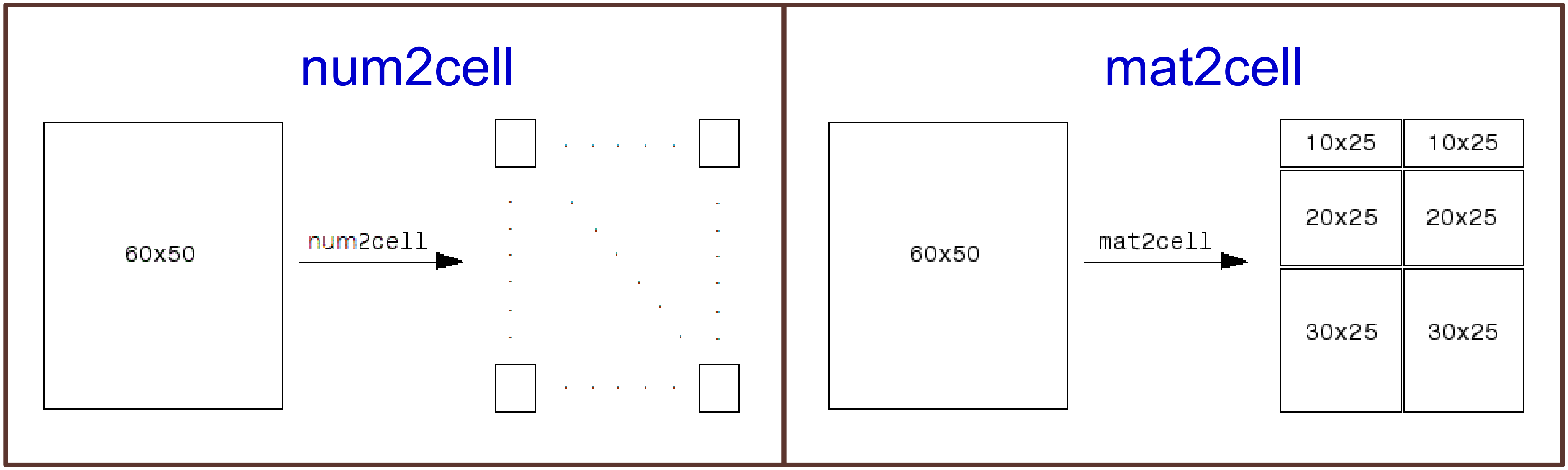
# Cell Array Functions

| | |
|---|---|
| cell | Create cell array |
| cell2mat | Convert cell array to numeric array |
| cell2struct | Convert cell array to structure array |
| celldisp | Cell array contents |
| cellfun | Apply function to each cell in cell array |
| cellplot | Graphically display structure of cell array |
| cellstr | Create cell array of strings from character array |
| iscell | Determine whether input is cell array |
| mat2cell | Convert array to cell array with different sized cells |
| num2cell | Convert array to cell array with consistently sized cells |
| struct2cell | Convert structure to cell array |

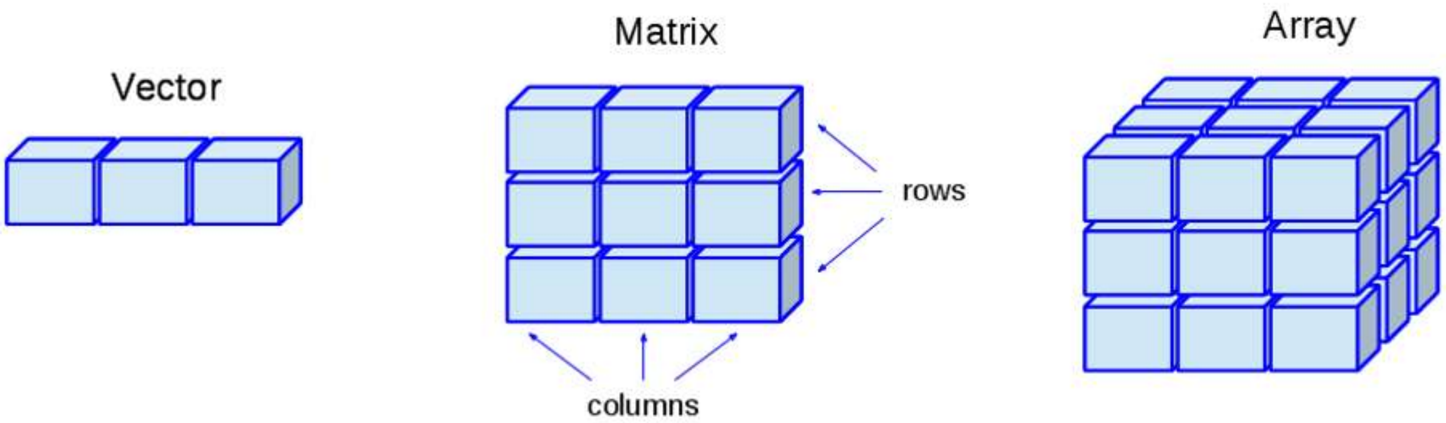
# num2cell() and mat2cell()

- Transform a matrix into a cell variable

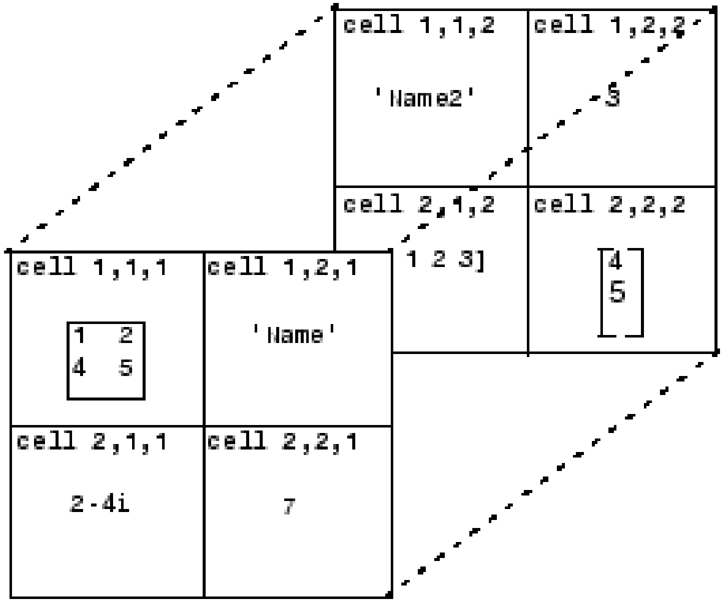
```
a = magic(3)
b = num2cell(a)
c = mat2cell(a, [1 1 1], 3)
```



# Multidimensional Array



```
A{1,1,1} = [1 2;4 5];
A{1,2,1} = 'Name';
A{2,1,1} = 2-4i;
A{2,1,1} = 7;
A{1,1,2} = 'Name2';
A{1,2,2} = 3;
A{2,1,2} = 0:1:3;
A{2,2,2} = [4 5]';
```



# cat()

- Array concatenation

```
A=[1 2;3 4]; B=[5 6;7 8];
```

```
C=cat(1,A,B)
```

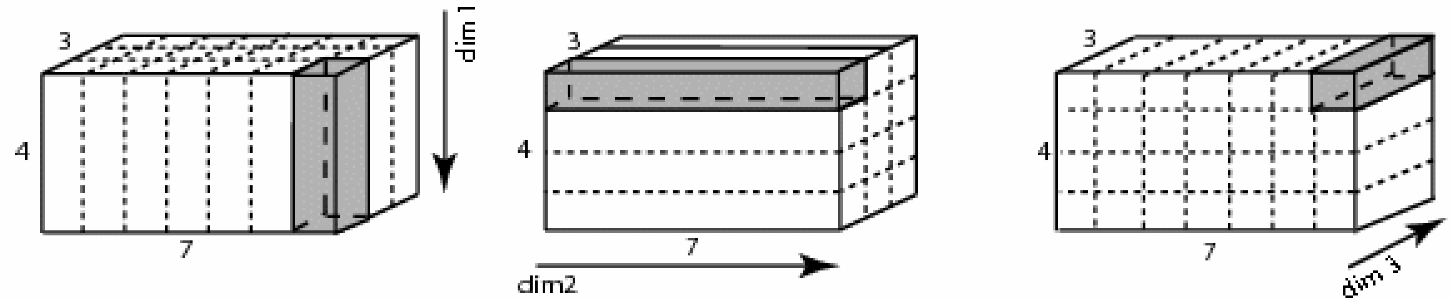
```
C=cat(2,A,B)
```

```
C=cat(3,A,B)
```

```
1  2
3  4
5  6
7  8
```

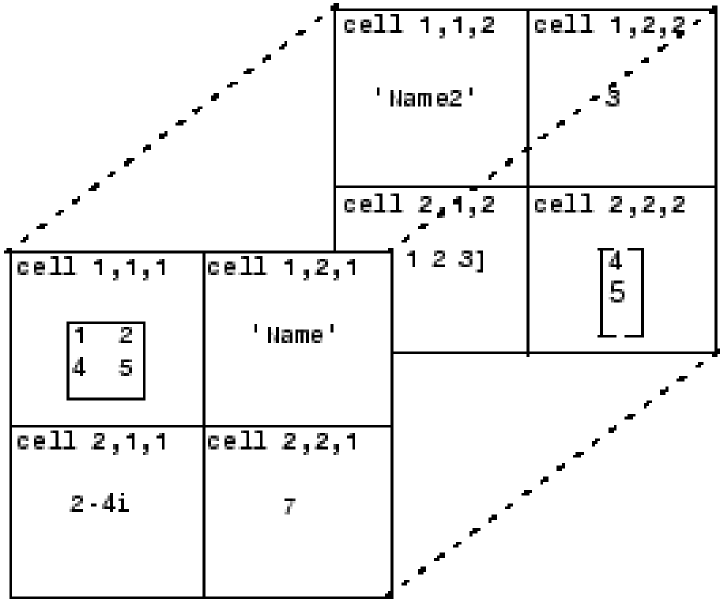
```
1  2  5  6
3  4  7  8
```

```
      5  6
      7  8
1  2
3  4
```



# Multidimensional Array

```
A{1,1} = [1 2;4 5];
A{1,2} = 'Name';
A{2,1} = 2-4i;
A{2,2} = 7;
B{1,1} = 'Name2';
B{1,2} = 3;
B{2,1} = 0:1:3;
B{2,2} = [4 5]';
C = cat(3, A, B)
```



# reshape()

- Returns a new array with assigned rows and columns

```
A = {'James Bond', [1 2;3 4;5 6]; pi, magic(5)}
C = reshape(A,1,4)
```

- Create a matrix B from the matrix A below using `reshape`:

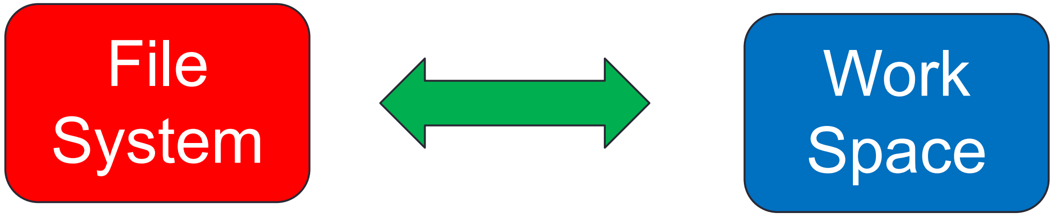
```
A = [1:3; 4:6];
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \Rightarrow B = \begin{bmatrix} 1 & 5 \\ 4 & 3 \\ 2 & 6 \end{bmatrix}$$

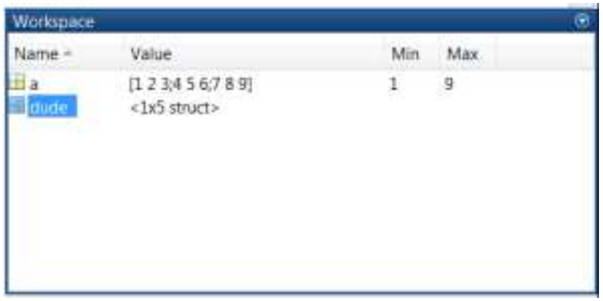
# Checking Variable And Variable Status

| | |
|---|---|
| `isinteger` | Determine if input is integer array |
| `islogical` | Determine if input is logical array |
| `isnan` | Detect an element that isnot a number (NaN) |
| `isnumeric` | Determine if input is numeric array |
| `isprime` | Detect prime elements of array |
| `isreal` | Determine if all array elements are real numbers |
| `iscell` | Determine if input is cell array |
| `ischar` | Determine if input is character array |
| `isempty` | Determine if input is empty array |
| `isequal` | Determine if arrays are numerically equal |
| `isfloat` | Determine if input is floating-point array |
| `isglobal` | Determine if input is global variable |
| `ishandle` | Detect valid graphics object handles |
| `isinf` | Detect infinite elements of array |

# File Access


File System

Work Space



| Name | Value | Min | Max |
|---|---|---|---|
| a | [1 2 3;4 5 6;7 8 9] | 1 | 9 |
| dude | <1x5 struct> | | |

- Supported file formats:

| File Content | Extension | Description | Import Function | Export Function |
|---|---|---|---|---|
| MATLAB formatted data | MAT | Saved MATLAB workspace | `load` | `save` |
| Text | | Space delimited numbers | `load` | `save` |
| Spreadsheet | XLS, XLSX | | `xlsread` | `xlswrite` |

# save() and load()

- Save (all) workspace data to a file:

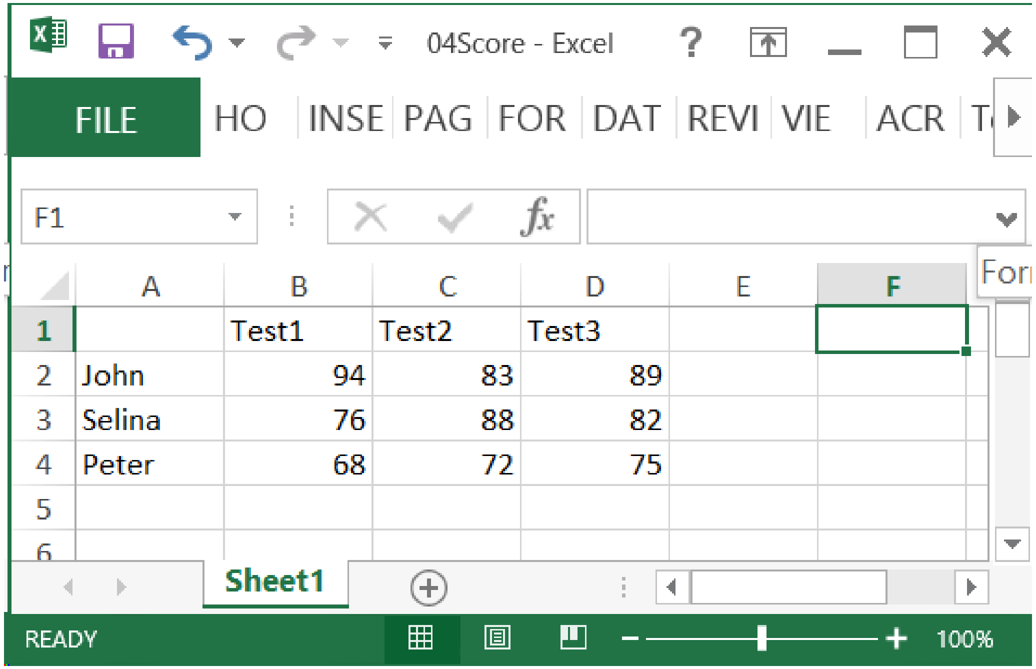
```
clear;  a = magic(4);
save mydata1.mat
save mydata2.mat -ascii
```

- Load data stored in a file:

```
load('mydata1.mat')
load('mydata2.mat','-ascii')
```

- How does one save a specific variable?

# Excel File Reading: `xlsread()`



|  | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 |  | Test1 | Test2 | Test3 |  |  |
| 2 | John | 94 | 83 | 89 |  |  |
| 3 | Selina | 76 | 88 | 82 |  |  |
| 4 | Peter | 68 | 72 | 75 |  |  |
| 5 |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |

- Read from Excel spreadsheet

```
Score = xlsread('04Score.xlsx')
Score = xlsread('04Score.xlsx', 'B2:D4')
```

# Excel File Writing: `xlswrite()`

- Calculate the means and write into Excel spreadsheet

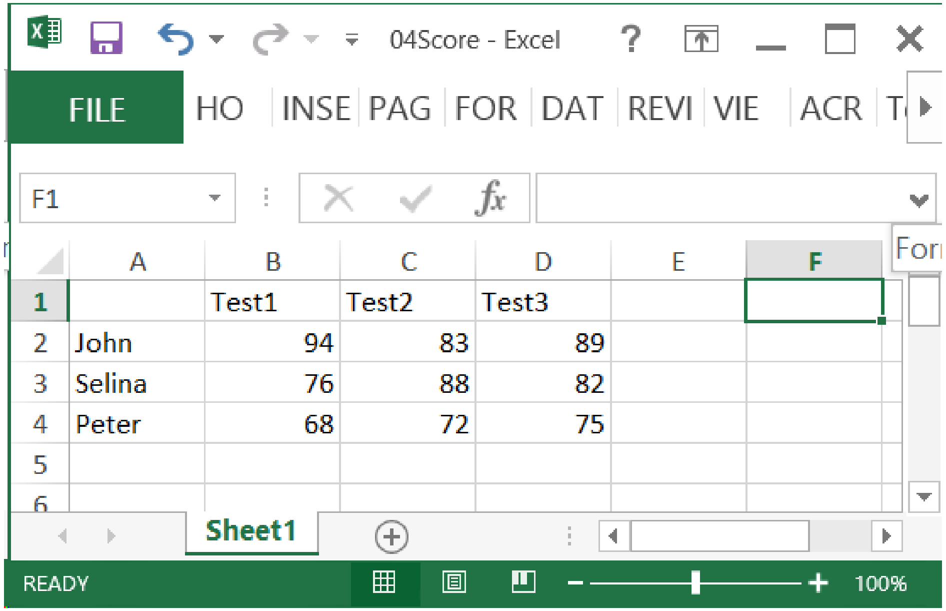
```
M = mean(Score')';
xlswrite('04Score.xlsx', M, 1, 'E2:E4');
xlswrite('04Score.xlsx', {'Mean'}, 1, 'E1');
```

- Calculate the standard deviations and write them into column F

# Getting Text in Excel Spreadsheet

- Getting both the text and numbers

```
[Score Header] = xlsread('04Score.xlsx')
```

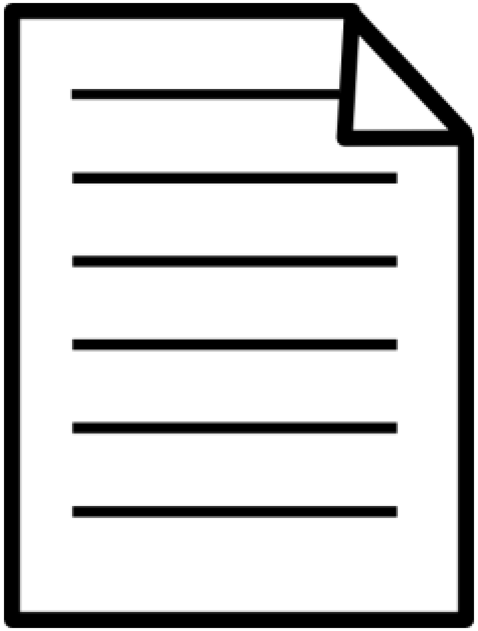


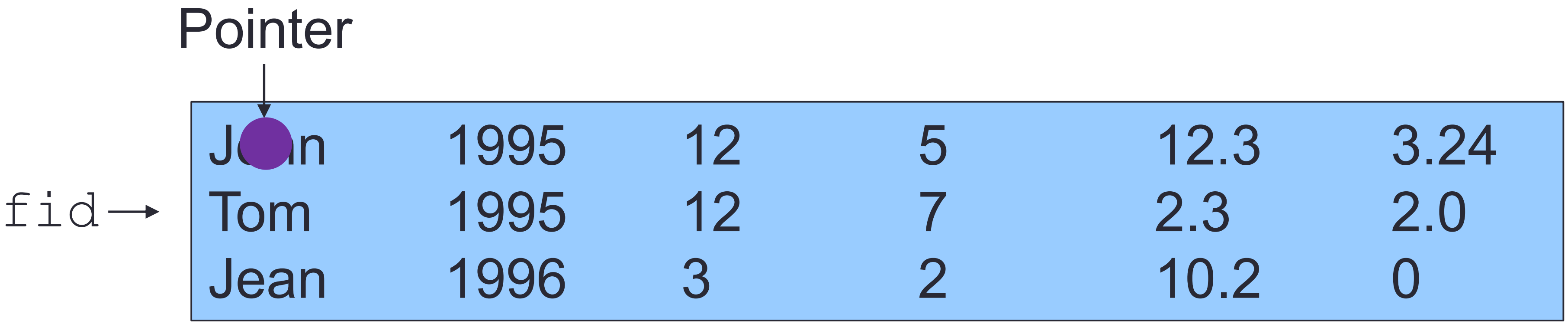
| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | | Test1 | Test2 | Test3 | | |
| 2 | John | 94 | 83 | 89 | | |
| 3 | Selina | 76 | 88 | 82 | | |
| 4 | Peter | 68 | 72 | 75 | | |
| 5 | | | | | | |
| 6 | | | | | | |

- How do we write both the text and number into an Excel file?

# Low-level File Input/Output

- Read and write file at the byte or character level
- A file has an ID `fid`
- Location in the file is specified by a pointer that can be moved around





| | | | | | |
|---|---|---|---|---|---|
| J●n | 1995 | 12 | 5 | 12.3 | 3.24 |
| Tom | 1995 | 12 | 7 | 2.3 | 2.0 |
| Jean | 1996 | 3 | 2 | 10.2 | 0 |

# Low-level File I/O Functions

| Function | Description |
|---|---|
| `fopen` | Open file, or obtain information about open files |
| `fclose` | Close one or all open files |
| `fscanf` | Read data from text file |
| `fprintf` | Write data to text file |
| `feof` | Test for end-of-file |

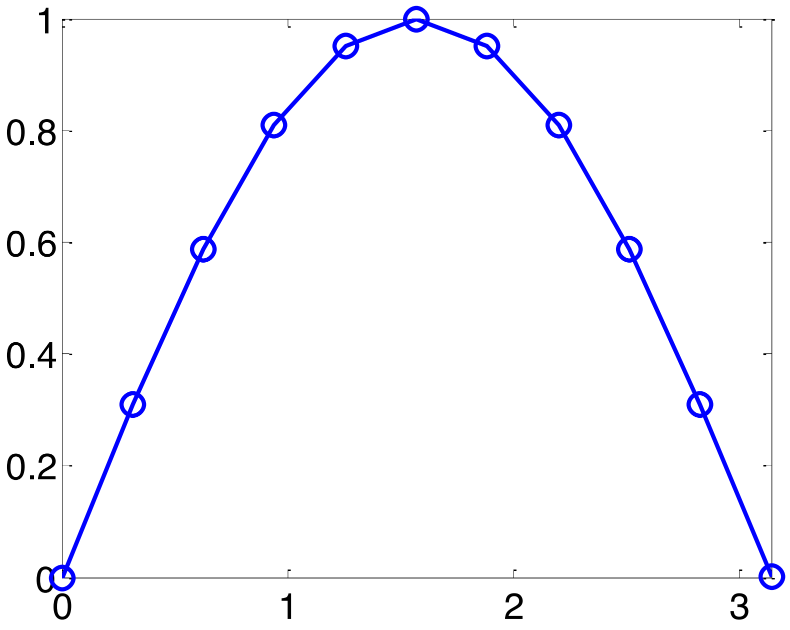
- Open and close a file:

```
fid = fopen('[filename]', '[permission]');
```

```
'r'     'r+'    'w'
'w+'    'a'     'a+'
```

```
status = fclose(fid);
```

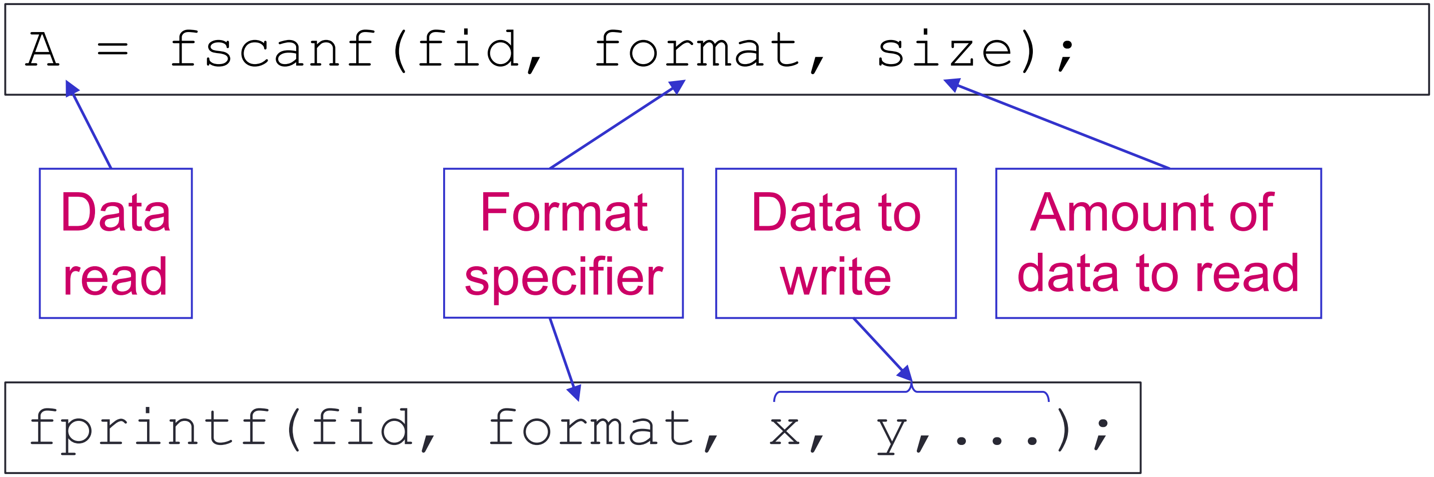
# Writing Sine Values into A File



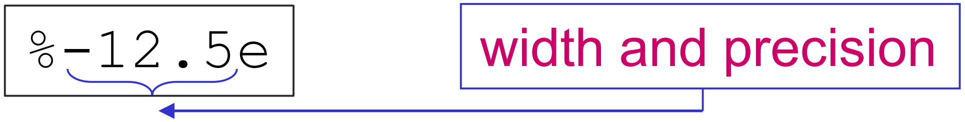
| x | y |
|---|---|
| 0.000 | 0.0000 |
| 0.314 | 0.3090 |
| 0.628 | 0.5878 |
| 0.942 | 0.8090 |
| 1.257 | 0.9511 |
| 1.571 | 1.0000 |
| 1.885 | 0.9511 |
| 2.199 | 0.8090 |
| 2.513 | 0.5878 |
| 2.827 | 0.3090 |
| 3.142 | 0.0000 |

```
x = 0:pi/10:pi; y = sin(x);  fid = fopen('sinx.txt','w');
for i=1:11
    fprintf(fid,'%5.3f %8.4f\n', x(i), y(i));
end
fclose(fid);  type sinx.txt
```

# Read and Write through Formatted I/O



- Read: `A = fscanf(fid, format, size);`
- Write: `fprintf(fid, format, x, y,...);`
- Format specifier: `%-12.5e`



| Specifier | Description |
|---|---|
| `%c` | Single character |
| `%d` | Decimal notation (signed) |
| `%e` | Exponential notation |
| `%f` | Fixed-point notation |
| `%g` | The more compact of `%e` or `%f` |
| `%o` | Octal notation (unsigned) |
| `%s` | String of characters |
| `%u` | Decimal notation (unsigned) |
| `%x` | Hexadecimal notation |

# Reading from Files

- Check if it is the end of file: `feof(fid)`
- 04asciiData.txt:

```
John    1995    12    5    12.3    3.24
Tom     1995    12    7    2.3     2.0
Jean    1996    3     2    10.2    0
```

```
fid = fopen('04asciiData.txt','r'); i = 1;
while ~feof(fid)
  name(i,:) = fscanf(fid,'%5c',1);
  year(i)   = fscanf(fid,'%d',1);
  no1(i)    = fscanf(fid,'%d',1);
  no2(i)    = fscanf(fid,'%d',1);
  no3(i)    = fscanf(fid,'%g',1);
  no4(i)    = fscanf(fid,'%g\n');
  i=i+1;
end
fclose(fid);
```

# End of Class