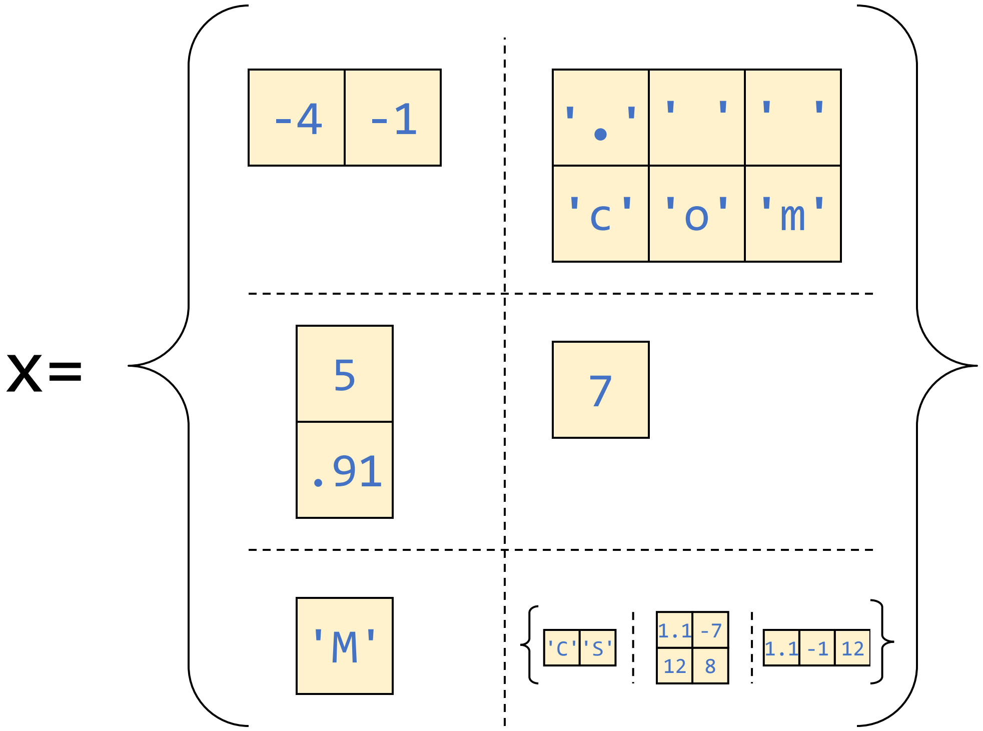
- Previous Lecture:
  - Review Linear Search
  - Cell arrays
- Today's Lecture:
  - File input/output
  - Using built-in function `sort`
  - Motivating packaging
- Announcements:
  - Complete Prelim 2 logistics survey on Canvas by Sun
  - Start reviewing for Prelim 2 (bring questions to review session Wed)
  - Tutoring appointments, office hours available
  - Look for P5 tomorrow; start early!

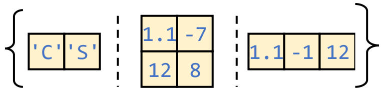
Review: cell arrays



Nested, heterogeneous

- x{3,1} → 'M'

- x{1,1} → [-4 -1]
- x{1,1}(2) → -1

- x{3,2} → 

- X{3,2}{1} → 'CS'
- X{3,2}{1}(2) → 'S'

I want to put in the 3rd cell of cell array C a **`char`** row vector. Which is correct?

A. `C{3} = 'a cat';`

B. `C{3} = ['a ' 'cat'];`

C. `C(3) = {'a ' 'cat'};`

D. Two answers above are correct

E. Answers A, B, C are all correct

Review question

Given the cell array:

```
x= { 'A', [3, 1, 4], uint8(zeros(6,4)) }
```

Which expression changes the 1 in x to a 5?

A
```
x(2,2)= 5
```

B
```
y= x{2};
y(2)= 5
```

C
```
x{2}(2)= 5
```

D
```
x(2)= [3, 5, 4]
```

A detailed sort-a-file example

File `statePop.txt` contains state population data sorted alphabetically by state. Create a new file

`statePopSm2Lg.txt`

that is structured the same as `statePop.txt` *except that the states are ordered from smallest to largest according to population.*

statePop.txt

```
Alabama          4557808
Alaska            663661
Arizona          5939292
Arkansas         2779154
California      36132147
Colorado         4665177
   :                 :
   :                 :
```

- Need the pop as *numbers* for sorting.
- Can't just sort the pop—have to maintain association with the state names.

First, read the file and store each line in a cell of a cell array

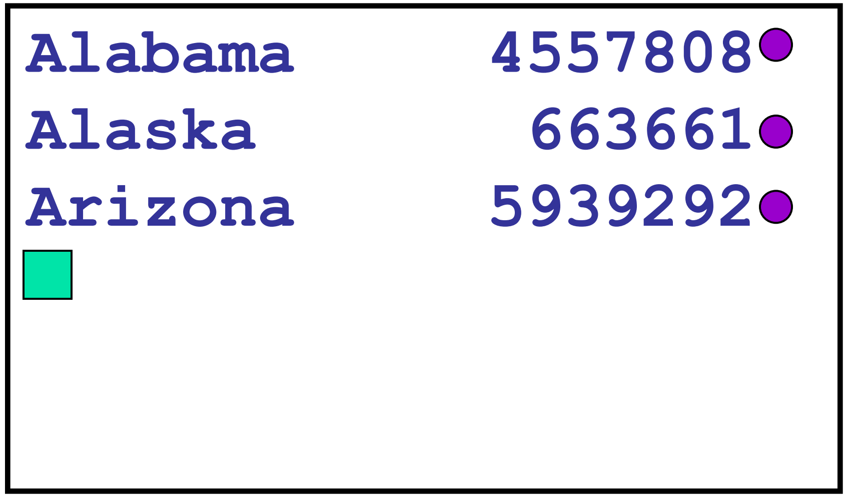
```
C = file2cellArray('StatePop.txt');
```

statePop.txt

```
Alabama          4557808
Alaska            663661
Arizona          5939292
Arkansas         2779154
California      36132147
Colorado         4665177
   :                 :
   :                 :
```

```
C= { 'Alabama              4557808';
     'Alaska                663661';
    …}
```

End-of-line and end-of-file



```
Alabama        4557808●
Alaska          663661●
Arizona        5939292●
🟩
```

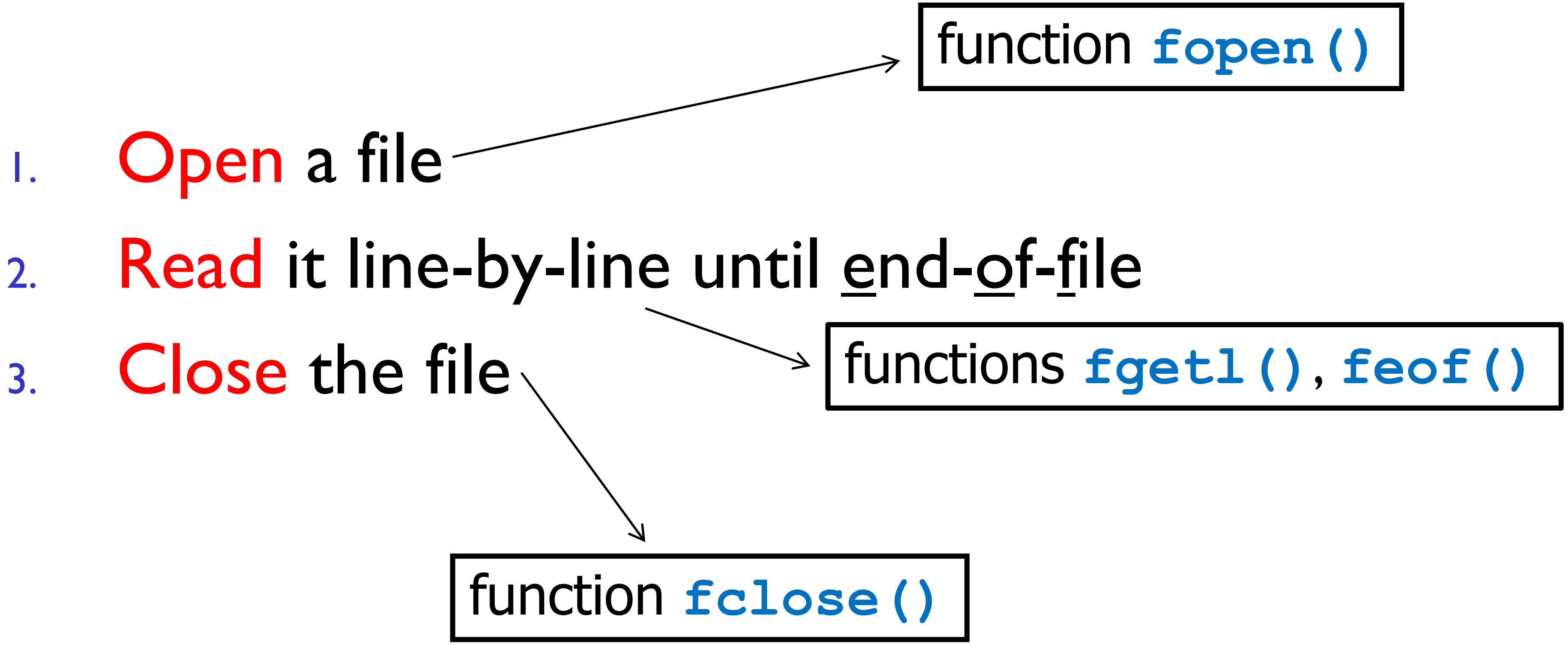
stateData.txt

● Line feed character ('\n') marks the end of a line

🟩 Computer knows how many characters are in file, and therefore where it ends.

eof stands for end of file

Read data from a file



1. Open a file
2. Read it line-by-line until end-of-file
3. Close the file

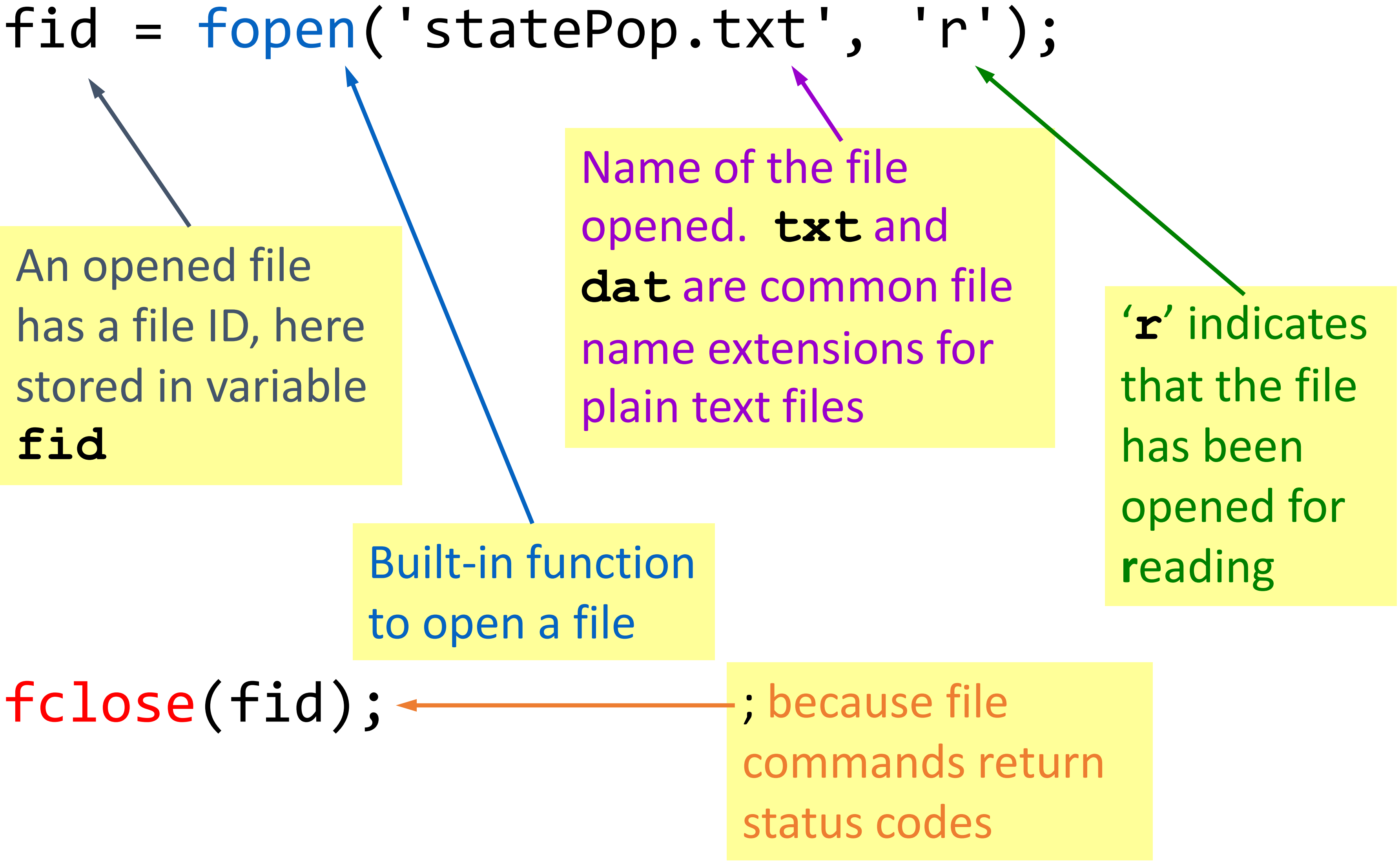
function `fopen()`

functions `fgetl()`, `feof()`

function `fclose()`

Closing a file is like the `end` keyword – need to tell MATLAB when you're done

1 & 3: Open (and close) file



```
fid = fopen('statePop.txt', 'r');
```

An opened file has a file ID, here stored in variable **`fid`**

Built-in function to open a file

Name of the file opened. **`txt`** and **`dat`** are common file name extensions for plain text files

'**`r`**' indicates that the file has been opened for **r**eading

```
fclose(fid);
```

; because file commands return status codes

2: Read each line and store it in cell array

```
fid = fopen('statePop.txt', 'r');

k= 0;
while ~feof(fid)
    k= k+1;
    Z{k}= fgetl(fid);
end

fclose(fid);
```

False until end-of-file is reached

🐞 Doesn't work for empty files

Get the next line.
(Each call gets one line; you cannot go to a specific line.)

```
function CA = file2cellArray(fname)
% fname is a string that names a non-empty
%   file in the current directory.
% CA is a cell array with CA{k} being the
%   k-th line in the file.

fid= fopen(fname, 'r');
k= 0;
while ~feof(fid)
   k= k+1;
   CA{k}= fgetl(fid);
end
fclose(fid);
```

C

$$\left\{\begin{array}{ll} \text{'Alab} & \text{4558000'} \\ \text{'Alas} & \text{664000'} \\ \vdots & \\ \text{'Cali} & \text{36132000'} \\ & \\ \text{'Verm} & \text{623000'} \\ \vdots & \\ \text{'Wyom} & \text{509000'} \end{array}\right\}$$

cell array
of strings
in alpha. order

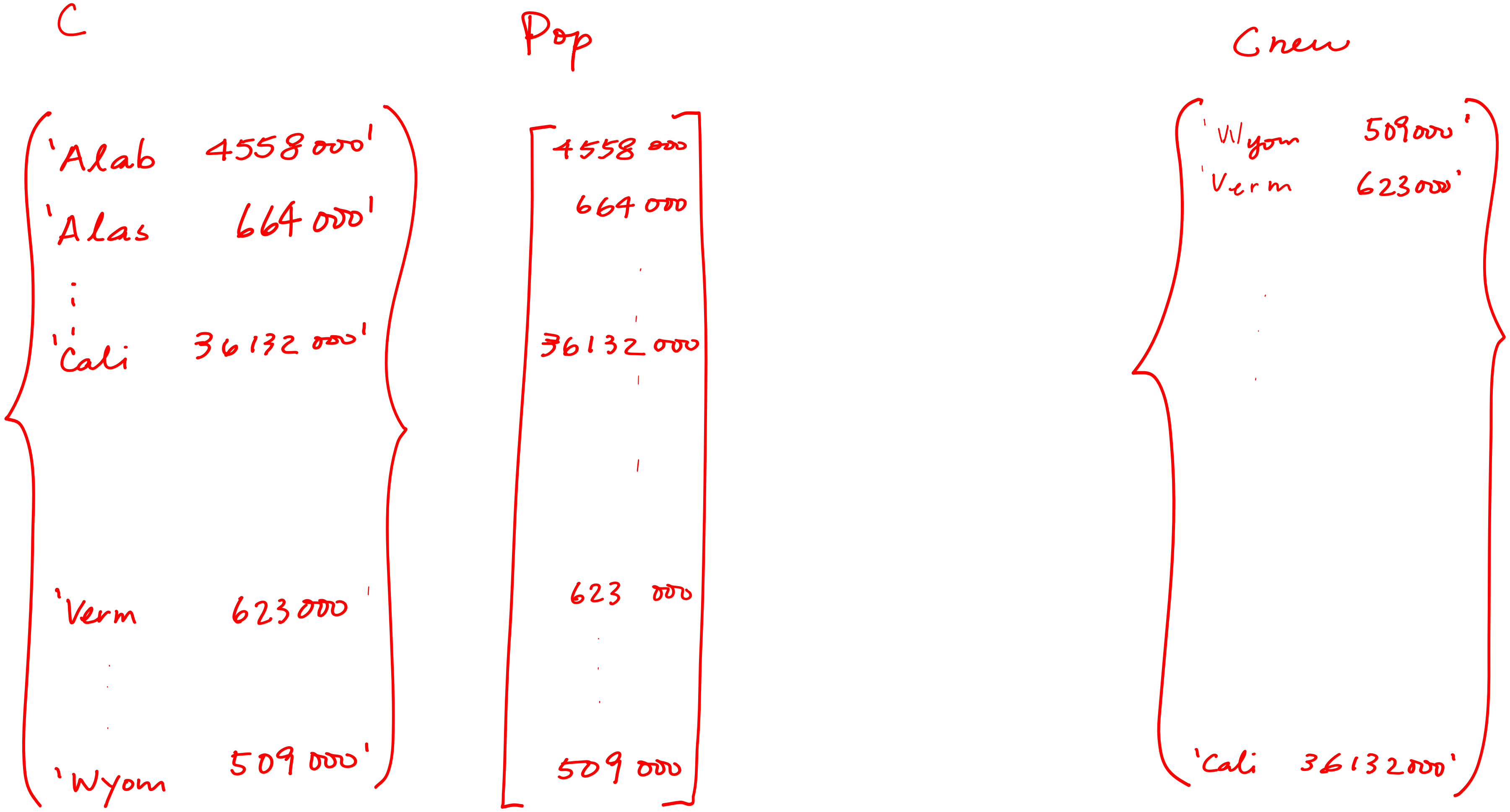
C

```
{ 'Alab   4558000'
  'Alas    664000'
    :
  'Cali  36132000'


  'Verm    623000'
    :
  'Wyom    509000' }
```

cell array
of strings
in alpha. order

Cnew

```
{ 'Wyom    509000'
  'Verm    623000'
    :

  'Cali  36132000' }
```



cell array
of strings
in alpha. order

vector
of numbers

Extracting population

- Two steps:
  1. Extract substring containing pop (and not name)
  2. Convert string (char vector) into number (scalar)

```
New York        19254630
North Carolina   8683242
123456789012345678901234
         1         2
```

Slicing question

Assume ‘statePop.txt’ is read into C using `file2CellArray()`. Which of these expressions evaluates to ‘zona’?

`statePop.txt`

```
Alabama              4557808
Alaska                663661
Arizona              5939292
Arkansas             2779154
California          36132147
Colorado             4665177
    :                    :
    :                    :
```

A `C{3,4:7}`

B `C(3,4:7)`

C `C{3}(4:7)`

D `C(4:7,3)`

Next, get the populations into a numeric vector

```
C = file2cellArray('StatePop.txt');
n = length(C);
pop = zeros(n,1);
for i=1:n
   S = C{i};
   pop(i) = str2double(S(16:24));
end
```

Converts a *string* representing a numeric value (digits, decimal point, spaces) to the numeric value → scalar of type *double*. E.g., `x=str2double(’  -3.24   ’)` assigns to variable `x` the numeric value -3.2400…

C

'Alab 4558000'
'Alas 664000'
⋮
'Cali 36132000'
'Verm 623000'
⋮
'Wyom 509000'

cell array
of strings
in alpha.order

Pop

4558000
664000
⋮
36132000
⋮
623000
⋮
509000

vector
of numbers

Cnew

'Wyom 509000'
'Verm 623000'
⋮
'Cali 36132000'

$$\begin{array}{ccccc}
C & Pop & s & idx & Cnew \\
\begin{Bmatrix} \text{'Alab} & \text{'} \\ \text{'Alas} & \text{664 000'} \\ \vdots & \\ \text{'Cali} & \text{36132 000'} \\ & \\ & \\ \text{'Verm} & \text{623000'} \\ \vdots & \\ \text{'Wyom} & \text{509 000'} \end{Bmatrix}
&
\begin{bmatrix} 4558\ 000 \\ 664\ 000 \\ \vdots \\ 36132\ 000 \\ \vdots \\ \\ 623\ 000 \\ \vdots \\ 509\ 000 \end{bmatrix}
&
\begin{bmatrix} 509000 \\ 623000 \\ \vdots \\ \\ \\ \\ \\ \\ 36132000 \end{bmatrix}
&
\begin{bmatrix} 50 \\ 45 \\ \vdots \\ \\ \\ \\ \\ \\ 5 \end{bmatrix}
&
\begin{Bmatrix} \text{'Wyom} & \text{509000'} \\ \text{'Verm} & \text{623000'} \\ \vdots & \\ & \\ & \\ & \\ & \\ & \\ \text{'Cali} & \text{36132000'} \end{Bmatrix}
\end{array}$$

cell array of strings in alpha. order

vector of numbers

vector of indices (ranks)

Built-In function `sort`

Syntax: `[y,idx] = sort(x)`

x:	10	20	5	90	15
y:	5	10	15	20	90
idx:	3	1	5	2	4

```
y(1) = x(3) = x(idx(1))
```

Built-In function `sort`

Syntax: `[y,idx] = sort(x)`

x:	10	20	5	90	15
y:	5	10	15	20	90
idx:	3	1	5	2	4

```
y(2) = x(1) = x(idx(2))
```

Built-In function `sort`

Syntax: `[y,idx] = sort(x)`

x:	10	20	5	90	15
y:	5	10	15	20	90
idx:	3	1	5	2	4

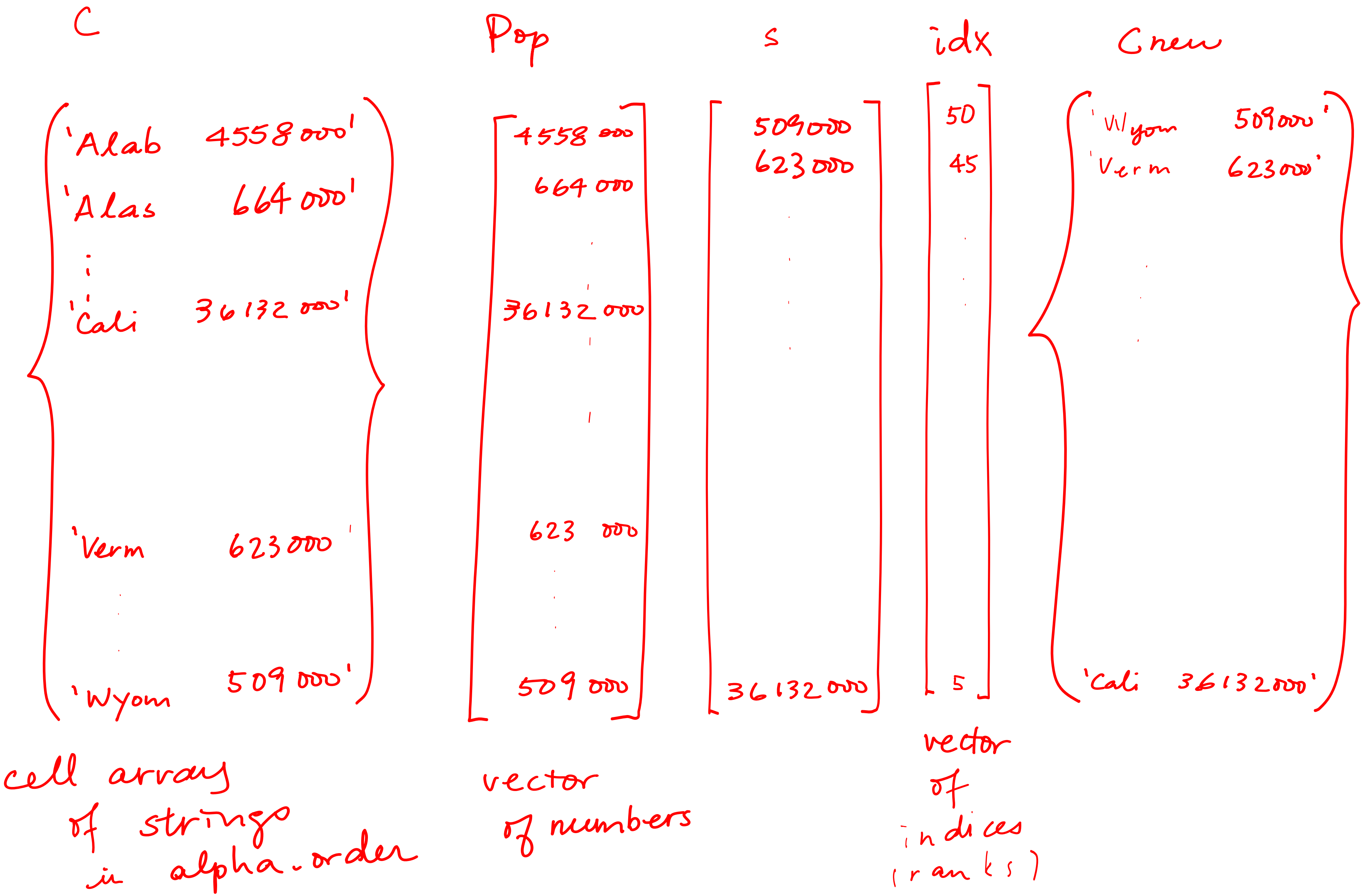
```
y(3) = x(5) = x(idx(3))
```

Built-In function `sort`

Syntax: `[y,idx] = sort(x)`

x:	10	20	5	90	15
y:	5	10	15	20	90
idx:	3	1	5	2	4

```
y(k) = x(idx(k))
```


C
Pop
s
idx
Cnew
'Alab 4558000'
'Alas 664000'
'Cali 36132000'
'Verm 623000'
'Wyom 509000'
4558000
664000
36132000
623000
509000
509000
623000
36132000
50
45
5
'Wyom 509000'
'Verm 623000'
'Cali 36132000'
cell array of strings in alpha. order
vector of numbers
vector of indices (ranks)

Sort from little to big

```
% C is cell array read from statePop.txt
% pop is vector of state pop (numbers)
[s,idx] = sort(pop);
Cnew = cell(length(C),1);
for i=1:length(Cnew)
    ithSmallest = idx(i);
    Cnew{i} = C{ithSmallest};
end
```

```
Cnew{i} = C{idx(i)};
```

Cnew

Wyoming	509294
Vermont	623050
North Dakota	636677
Alaska	663661
South Dakota	775933
Delaware	843524
Montana	935670
:	:
:	:
Illinois	12763371
Florida	17789864
New York	19254630
Texas	22859968
California	36132147

Sorting question

Assume you have `C`, `pop`, `s`, and `idx` as defined previously in this lecture. Write a code snippet that prints the names of the states whose populations are between the 20th and 40th percentile.

Statistics review: 1/5 of states will have smaller populations than the ones you print, and 3/5 of states will have larger populations.

Save results

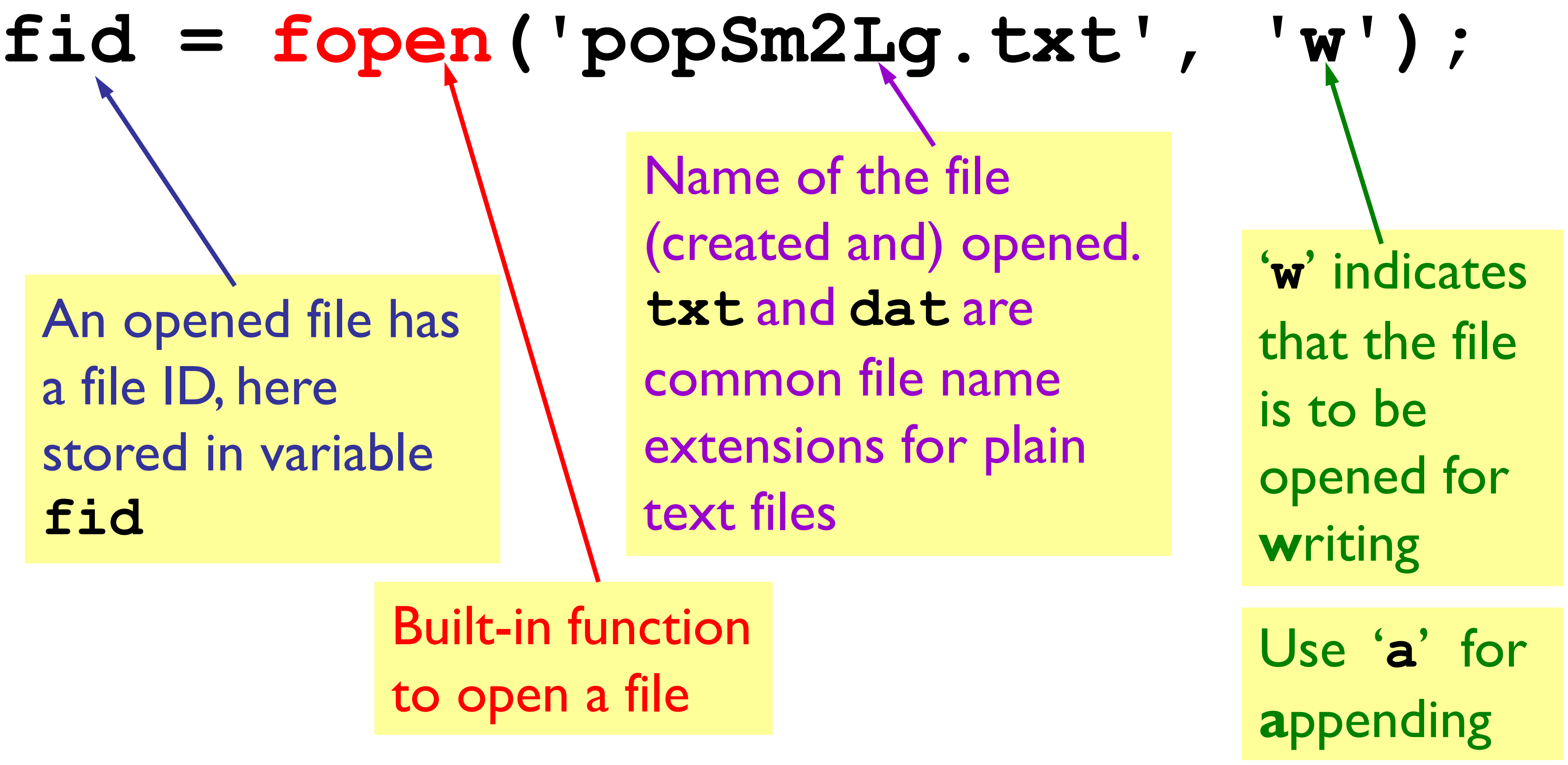
```
% C is cell array read from statePop.txt
% pop is vector of state pop (numbers)
[s,idx] = sort(pop);
Cnew = cell(length(C),1);
for i=1:length(Cnew)
    ithSmallest = idx(i);
    Cnew{i} = C{ithSmallest};
end
```

```
cellArray2file(Cnew,'statePopSm2Lg.txt')
```

A 3-step process to
read data from a file or
write data to a file

1. (Create and) open a file
2. Read data from or write data to the file
3. Close the file

1. Open a file (don't forget to later close the file)



```
fid = fopen('popSm2Lg.txt', 'w');
```

An opened file has a file ID, here stored in variable **`fid`**

Built-in function to open a file

Name of the file (created and) opened. **`txt`** and **`dat`** are common file name extensions for plain text files

'**`w`**' indicates that the file is to be opened for **writing**

Use '**`a`**' for **appending**

```
fclose(fid);
```

2. Write (print) to the file

```
fid = fopen('popSm2Lg.txt', 'w');

for i=1:length(Cnew)
    fprintf(fid, '%s\n', Cnew{i});
end

fclose(fid);
```

Printing is to be done to the file with ID `fid`

Substitution sequence specifies the string format (followed by a new-line character)

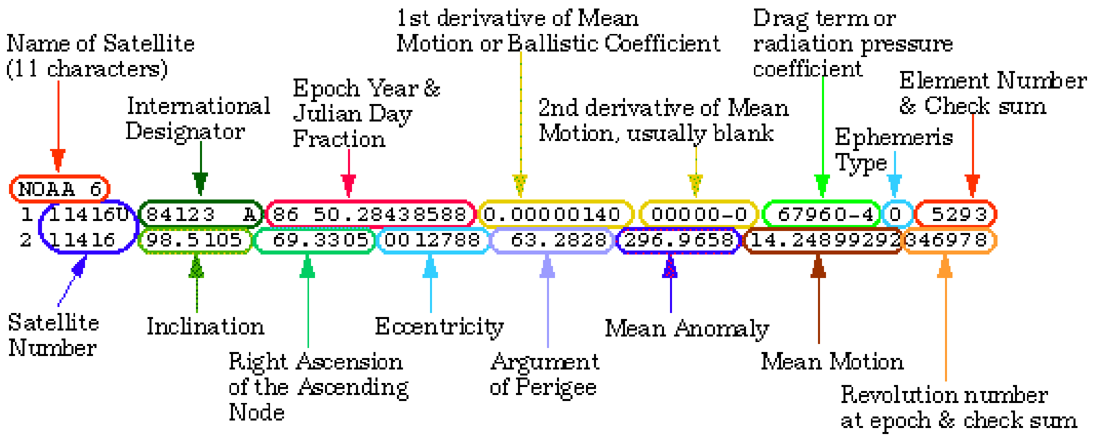
The **`i`**th *item in cell array* **`Cnew`**

```
function cellArray2file(CA, fname)
% CA is a cell array of strings.
% Create a file with the name
% specified by the string fname.
% The i-th line in the file is CA{i}

fid= fopen(fname, 'w');
for i= 1:length(CA)
    fprintf(fid, '%s\n', CA{i});
end
fclose(fid);
```

Storing only a selected (small) section of data from a big file

- The previous example reads the whole file and stores all the text
- If you're interested in only a small part of the data, storing everything is an overkill
- Read "issYear.m" posted on the website to learn how to store only the data that meet certain criteria

Example: NORAD two-line elements



```
NOAA 6
1 11416U 84123  A 86 50.28438588 0.00000140  00000-0  67960-4 0  5293
2 11416  98.5105  69.3305 0012788  63.2828 296.9658 14.24899292346978
```

```
ISS (ZARYA)
1 25544U 98067A   19280.43177083  .00000288  00000-0  13040-4 0  9993
2 25544  51.6437 164.6585 0007556 123.5429 237.5675 15.50172544192676
⋮
STARLINK-74
1 44293U 19029BL  19280.46307273  .00000774  00000-0  72445-4 0  9999
2 44293  53.0058 280.3384 0001435  93.2755 266.8397 15.05496611 21751
STARLINK-53
1 44294U 19029BM  19279.64653505  .00000628  00000-0  62400-4 0  9998
2 44294  52.9988 283.1290 0000873  99.6752 260.4335 15.05478127 19808
COSMOS 2534 [GLONASS-M]
1 44299U 19030A   19279.63973935  .00000042  00000-0  00000+0 0  9999
2 44299  64.7328 275.7191 0015277 282.8642  34.0841  2.13101948  2816
```

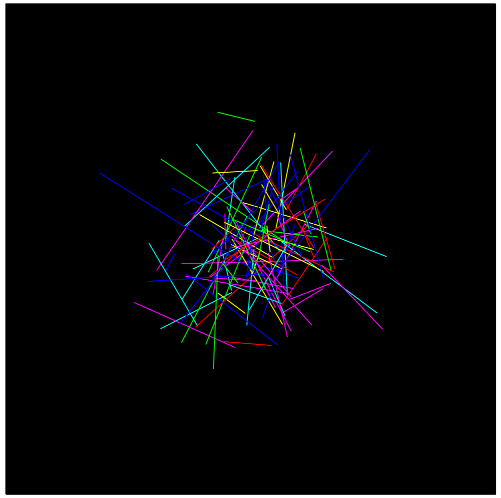
Website example: satellite launch year

1. Read line (satellite name)
2. While name is not ISS
    1. Read 2 lines (skip)
    2. Read line (satellite name)
3. Read line (record 1)
4. Extract characters 10 & 11
5. Convert to number, interpret as year

```
SCD 2
1 25504U 98060A   19288.18395014  .00000230  00000-0  13957-4 0  9992
2 25504  24.9967 317.5526 0017113 331.0386 103.7958 14.44077629107938
ISS (ZARYA)
1 25544U 98067A   19280.43177083  .00000288  00000-0  13040-4 0  9993
2 25544  51.6437 164.6585 0007556 123.5429 237.5675 15.50172544192676
⋮
STARLINK-53
1 44294U 19029BM  19279.64653505  .00000628  00000-0  62400-4 0  9998
2 44294  52.9988 283.1290 0000873  99.6752 260.4335 15.05478127 19808
COSMOS 2534 [GLONASS-M]
1 44299U 19030A   19279.63973935  .00000042  00000-0  00000+0 0  9999
2 44299  64.7328 275.7191 0015277 282.8642  34.0841  2.13101948  2816
```

Data are often related

- A point in the plane has an x coordinate and a y coordinate.
- If a program manipulates lots of points, there will be lots of x's and y's.
- Anticipate clutter. Is there a way to "package" the two coordinate values?



Packaging affects thinking

Our Reasoning Level:

P and Q are points. Compute the midpoint M of the connecting line segment.

Behind the scenes we do this:

$$M_x = (P_x + Q_x)/2$$
$$M_y = (P_y + Q_y)/2$$

We've seen this before: functions are used to "package" calculations.

This packaging (a type of abstraction) elevates the level of our reasoning and is critical for problem solving.