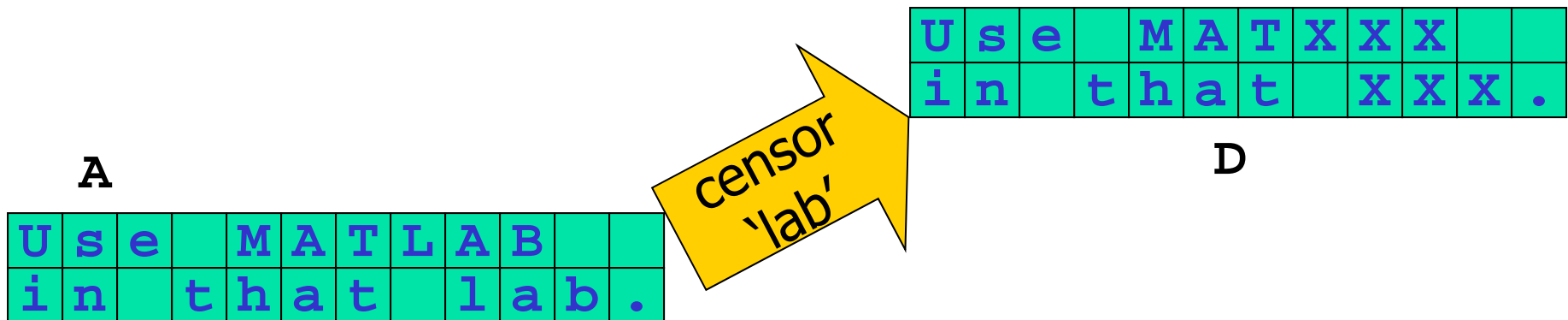


- Previous Lecture:
 - Characters and strings
- Today's Lecture:
 - More on characters and strings
 - Cell arrays
 - File input/output
- Announcement:
 - Discussion this week in computer lab

Example: censoring words

```
function D = censor(str, A)
% Replace all occurrences of string str in
% character matrix A with X's, regardless of
% case.
% Assume str is never split across two lines.
% D is A with X's replacing str.
```



```
function D = censor(str, A)
% Replace all occurrences of string str in character matrix A,
% regardless of case, with X's.
% A is a matrix of characters.
% str is a string. Assume that str is never split across two lines.
% D is A with X's replacing the censored string str.

D= A;
B= lower(A);
s= lower(str);
ns= length(str);
[nr,nc]= size(A);

% Build a string of X's of the right length

% Traverse the matrix to censor string str
```

```

function D = censor(str, A)
% Replace all occurrences of string str in character matrix A,
% regardless of case, with X's.
% A is a matrix of characters.
% str is a string. Assume that str is never split across two lines.
% D is A with X's replacing the censored string str.

```

Xs XXXXXXXXXX

```

D= A;
B= lower(A);
s= lower(str);
ns= length(str);
[nr,nc]= size(A);

```

	1	2	3	...	8	9	10	11	12			
B	u	s	e		m	a	t	l	a	b		
	i	n		t	h	a	t		l	a	b	.

```

% Build a string of X's of the right length

```

```

Xs= char( zeros(1,ns));
for k= 1:ns
    Xs(k)= 'X';
end

```

Returns an array of type double

Changes the type to char

```

% Traverse the matrix to censor string str

```

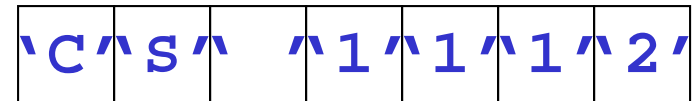
```

for r= 1:nr
    for c= 1:nc-ns+1
        if strcmp( s , B(r, c:c+ns-1) )==1
            D(r, c:c+ns-1)= Xs;
        end
    end
end
end

```

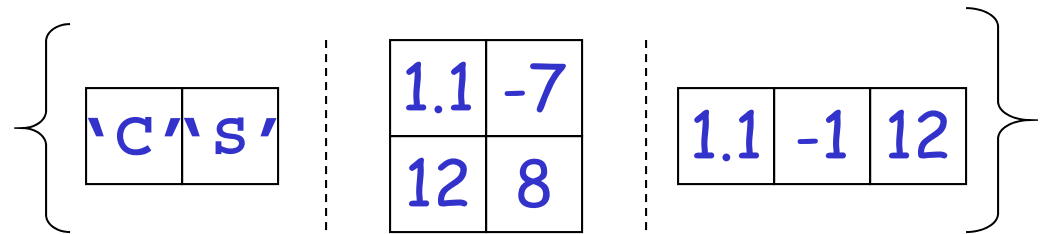
Array vs. Cell Array

■ Simple array



- Each component stores one scalar. E.g., one char, one double, or one uint8 value
- All components have the same type

■ Cell array



- Each cell can store something “bigger” than one scalar, e.g., a vector, a matrix, a string (vector of chars)
- The cells may store items of different types

1-d and 2-d examples ...

Vectors and matrices store values of the same type in all components

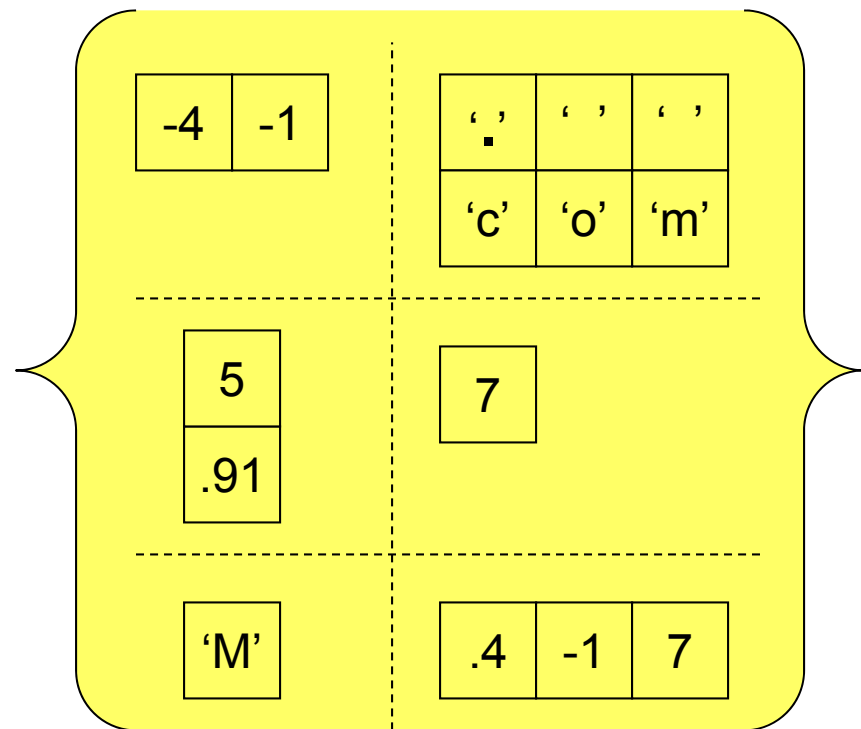
3.1
2
-1
9
1.1

5 x 1 matrix

'c'	'o'	'm'	' '	's'
'1'	'1'	'1'	'2'	' '
'M'	'a'	't'	' '	' '
' '	' '	'L'	'A'	'B'

4 x 5 matrix

Cell array: individual components may contain different types of data



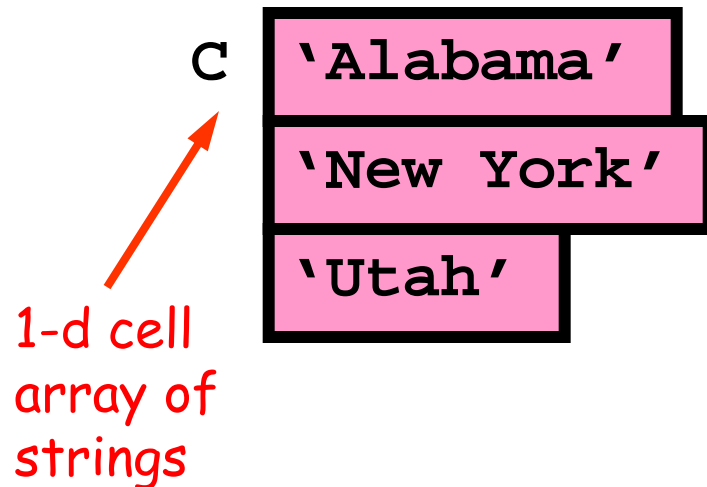
3 x 2 cell array

Cell Arrays of Strings

```
C = { 'Alabama', 'New York', 'Utah' }
```



```
C = { 'Alabama'; 'New York'; 'Utah' }
```



Contrast with
2-d array of characters

```
M = [ 'Alabama ' ; ...  
      'New York' ; ...  
      'Utah   ' ]
```

	'A'	'l'	'a'	'b'	'a'	'm'	'a'	' '
M	'N'	'e'	'w'	' '	'Y'	'o'	'r'	'k'
	'U'	't'	'a'	'h'	' '	' '	' '	' '

Use braces { } for creating and addressing cell arrays

Matrix

- Create

```
m = [ 5, 4 ; ...  
      1, 2 ; ...  
      0, 8 ]
```

- Addressing

```
m(2,1) = pi
```

Cell Array

- Create

```
C = { ones(2,2), 4 ; ...  
      'abc' , ones(3,1) ; ...  
      9 , 'a cell' }
```

- Addressing

```
C{2,1} = 'ABC'  
C{3,2} = pi  
disp(C{3,2})
```

Creating cell arrays...

```
C = {'Oct', 30, ones(3,2)};
```

is the same as

```
C = cell(1,3); % not necessary
```

```
C{1} = 'Oct';
```

```
C{2} = 30;
```

```
C{3} = ones(3,2);
```

You can assign the empty cell array: `D = {}`

Example: Represent a deck of cards with a cell array

`D{1} = 'A Hearts';`

`D{2} = '2 Hearts';`

`:`

`D{13} = 'K Hearts';`

`D{14} = 'A Clubs';`

`:`

`D{52} = 'K Diamonds';`

But we don't want to have to type all combinations of suits and ranks in creating the deck... How to proceed?

Make use of a suit array and a rank array ...

```
suit = { 'Hearts', 'Clubs', ...  
        'Spades', 'Diamonds' };  
  
rank = { 'A', '2', '3', '4', '5', '6', ...  
        '7', '8', '9', '10', 'J', 'Q', 'K' };
```

Then concatenate to get a card. E.g.,

```
str = [rank{3} \ ' suit{2} ];  
D{16} = str;
```

So D{16} stores '3 Clubs'


To get all combinations, use **nested loops**


```
i = 1; % index of next card

for k= 1:4
    % Set up the cards in suit k
    for j= 1:13
        D{i} = [ rank{j} ' ' suit{k} ];
        i = i+1;
    end
end
end
```

See function **CardDeck**

Example: deal a 12-card deck

D: 

N:  1, 5, 9 $4k-3$

E:  2, 6, 10 $4k-2$

S:  3, 7, 11 $4k-1$

W:  4, 8, 12 $4k$

```
% Deal a 52-card deck
```

```
N = cell(1,13); E = cell(1,13);
```

```
S = cell(1,13); W = cell(1,13);
```

```
for k=1:13
```

```
    N{k} = D{4*k-3};
```

```
    E{k} = D{4*k-2};
```

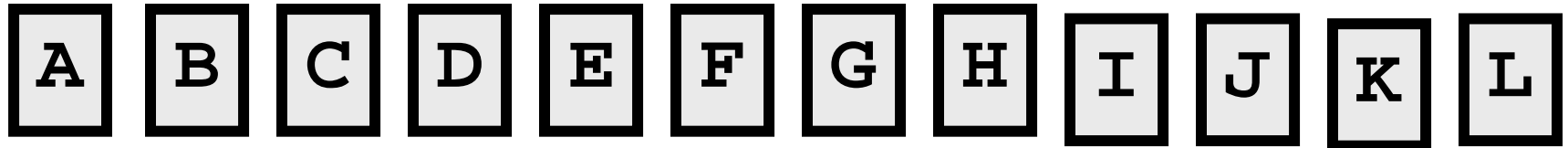
```
    S{k} = D{4*k-1};
```

```
    W{k} = D{4*k};
```

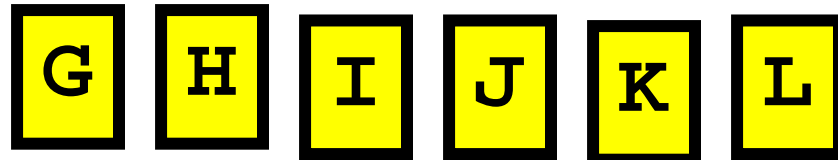
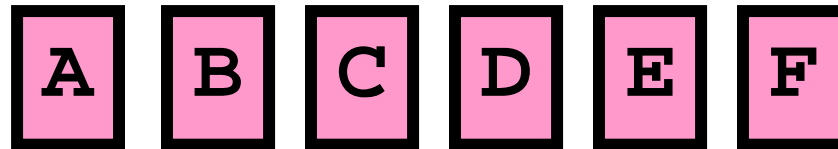
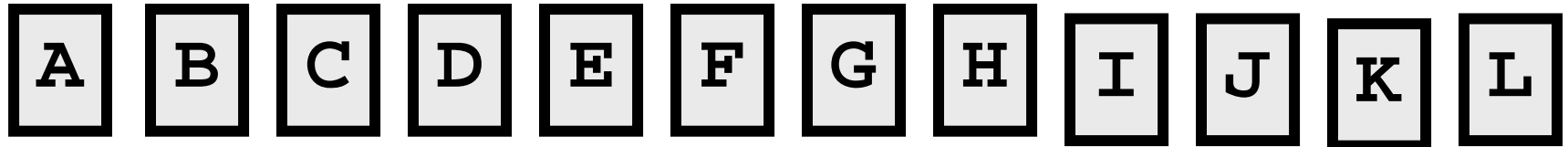
```
end
```

See function **Deal**

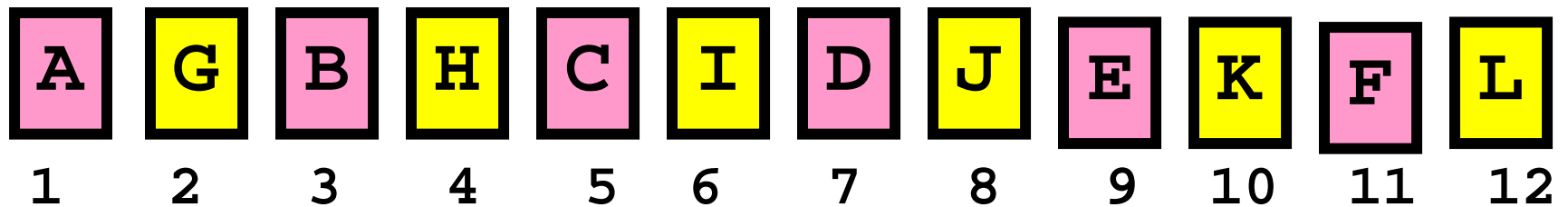
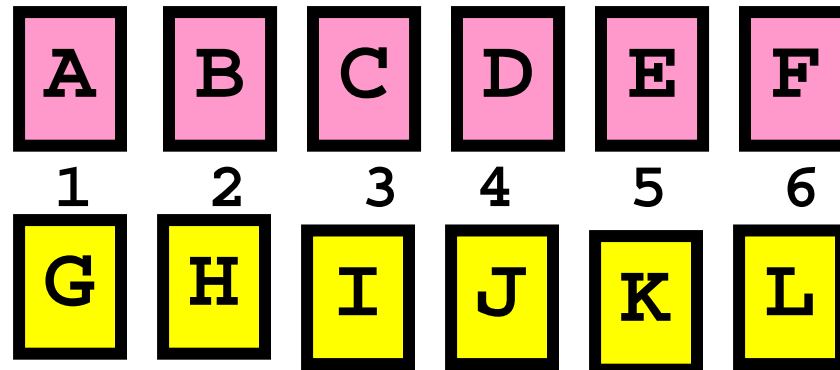
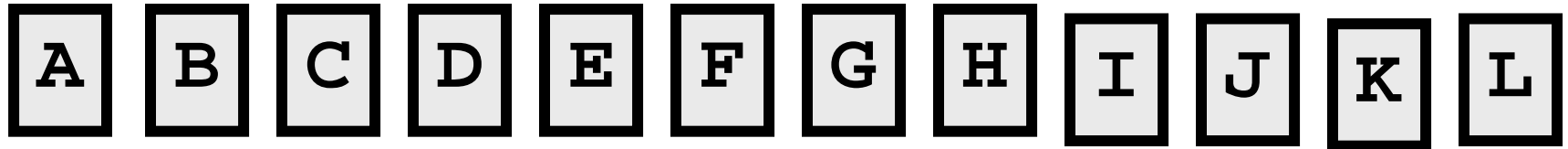
The “perfect shuffle” of a 12-card deck



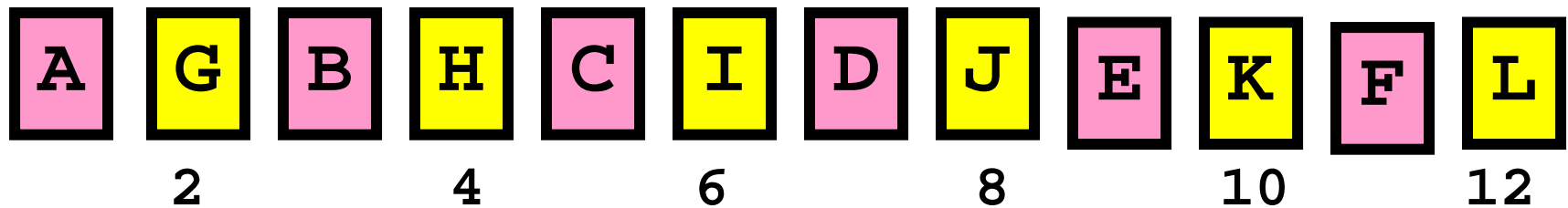
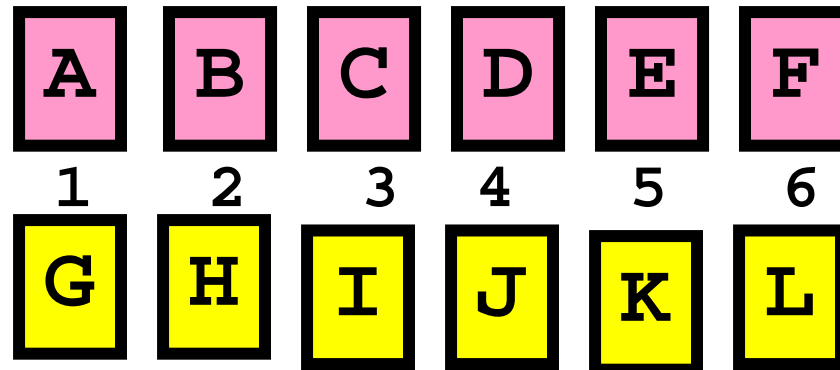
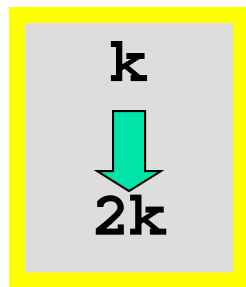
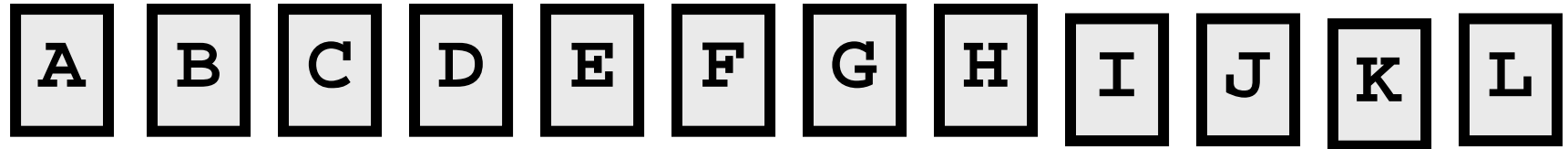
Perfect Shuffle, Step I: cut the deck



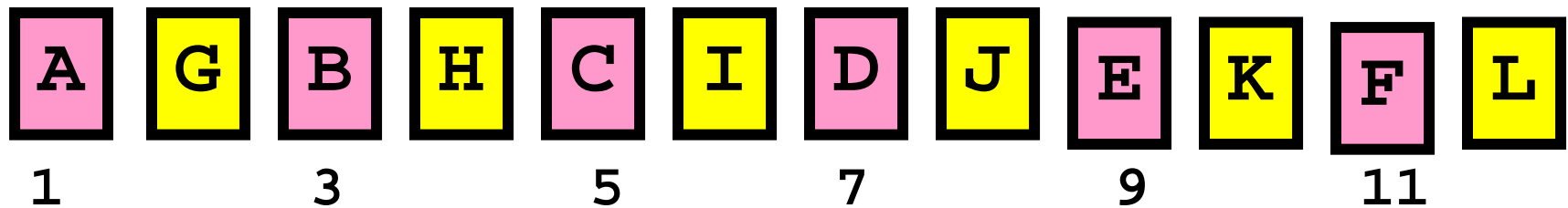
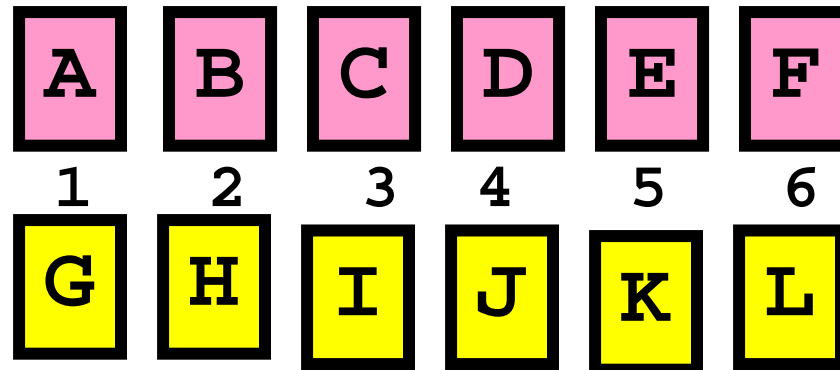
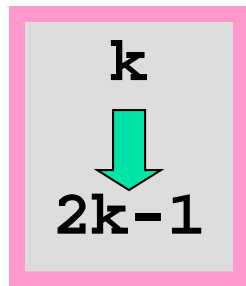
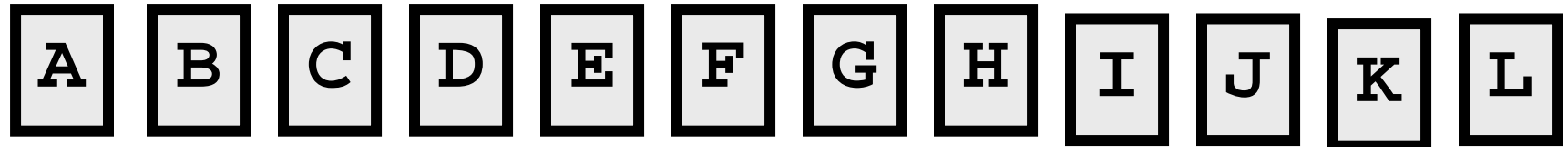
Perfect Shuffle, Step 2: Alternate



Perfect Shuffle, Step 2: Alternate

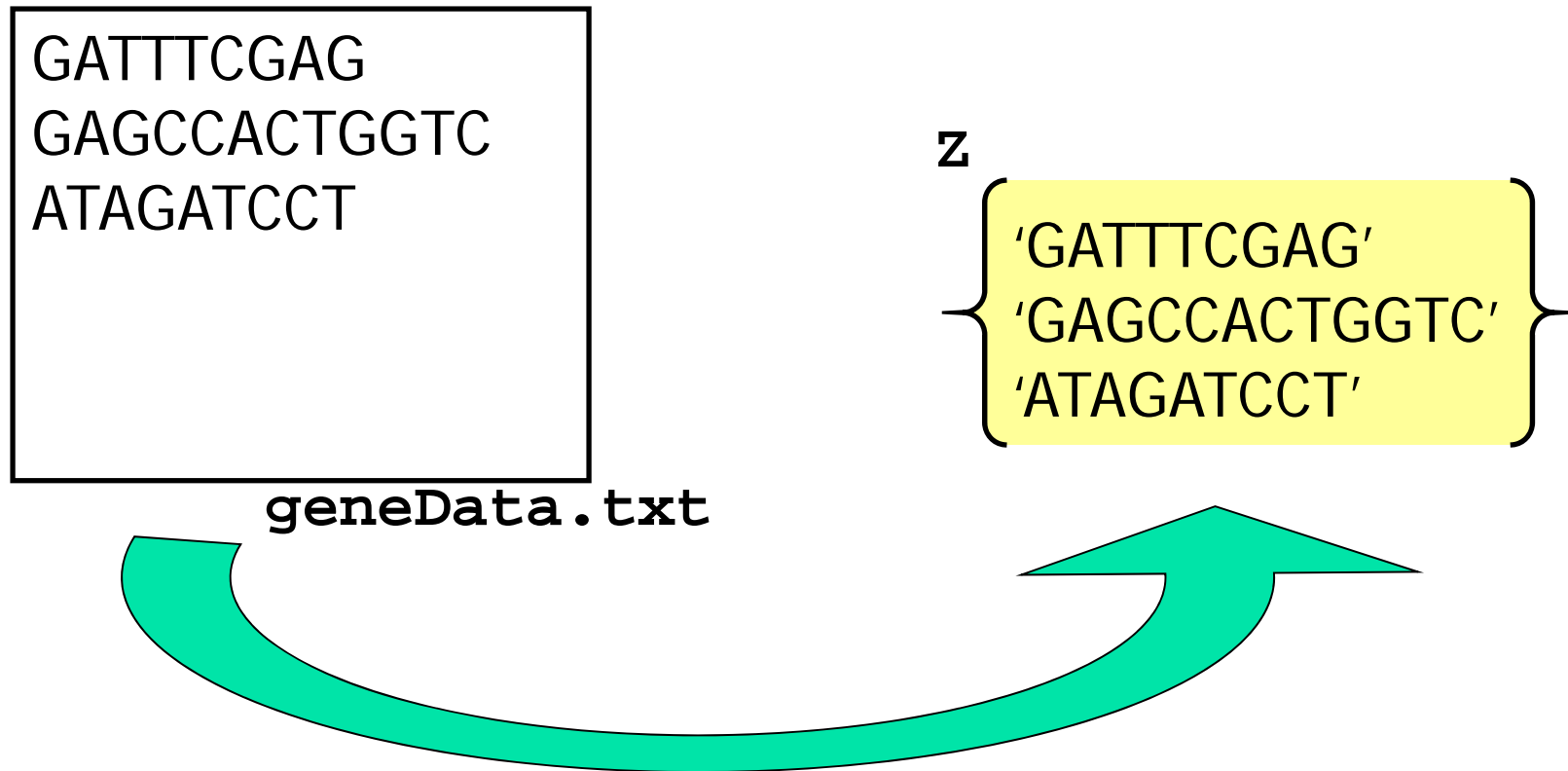


Perfect Shuffle, Step 2: Alternate



See function `shuffle`

Working with data files: Read the data in a file line-by-line and store the results in a cell array



How are lines separated?
How do we know when there are no more lines?

In a file there are hidden “markers”

```
GATTTCGAG ●  
GAGCCACTGGTC ●  
ATAGATCCT ●  
■
```

geneData.txt

- Carriage return marks the end of a line

- eof marks the end of a file

Read data from a file

1. **Open** a file
2. **Read** it line-by-line until eof
3. **Close** the file

I. Open the file

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

An open 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 reading

2. Read each line and store it in cell array

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

```
k = 0;
```

```
while ~feof(fid)
```

False until end-of-file is reached

```
    k = k + 1;
```

```
    z{k} = fgetl(fid);
```

```
end
```

Get the next line

3. Close the file

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

```
k= 0;
```

```
while ~feof(fid)
```

```
    k= k+1;
```

```
    z{k}= fgetl(fid);
```

```
end
```

```
fclose(fid);
```