- Previous Lecture:
  - Characters and strings
- Today's Lecture:
  - More on characters and strings
  - Cell arrays
  - File input/output
- Announcement:
  - Project 4 due tonight at 11pm. Late penalty reduced to 1 point for submission within 24 hrs after deadline
  - Prelim 2 on Nov 6th (Tues) at 7:30pm. Email Randy Hess (rbh27) ASAP about any conflict and include information on the conflicting event (course number, instructor name and email, etc.)

Example: removing all occurrences of a character

- From a genome bank we get a sequence
  ATTG CCG TA GCTA CGTACGC AACTGG AAATGGC CGTAT…
- First step is to "clean it up" by removing all the blanks. Write this function:

```
function s = removeChar(c, s)
% Return string s with all occurrences
% of character c removed
```

Example: removing all occurrences of a character

Can solve this problem using iteration—check one character (one component of the vector) at a time

```
function s = removeChar_loop(c, s)
% Return string s with all occurrences of
% character c removed.
```

Example: removing all occurrences of a character

Can solve this problem using iteration—check one character (one component of the vector) at a time

```
function s = removeChar_loop(c, s)
% Return string s with all occurrences of
% character c removed.

t= ’’;  % initialize empty string
for k= 1:length(s)




end
s= t;
```

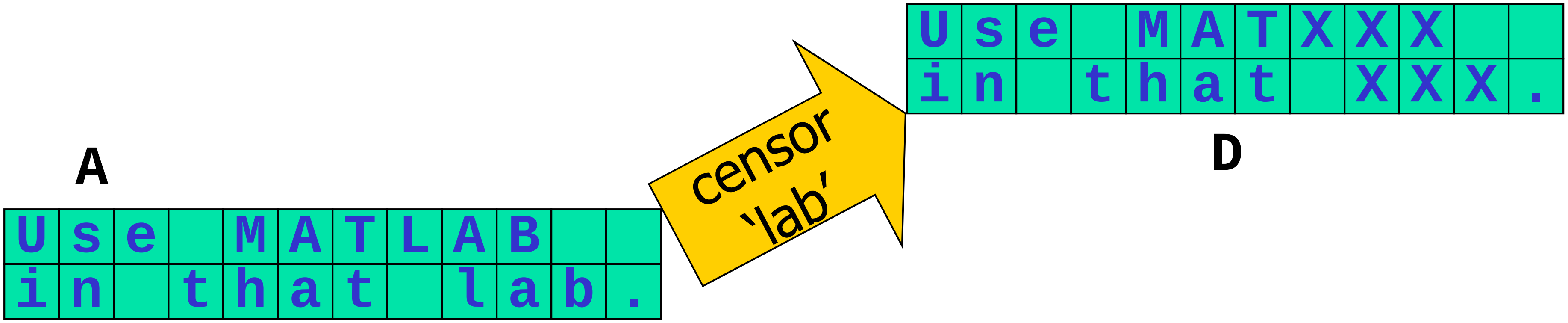
Example: removing all occurrences of a character

Can solve this problem using iteration—check one character (one component of the vector) at a time

```
function s = removeChar_loop(c, s)
% Return string s with all occurrences of
% character c removed.

t= ’’;  % initialize empty string
for k= 1:length(s)
    if s(k)~=c
        t= [t s(k)];
    end
end
s= t;
```

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.
```



A

U	s	e		M	A	T	L	A	B		
i	n		t	h	a	t		l	a	b	.

D

U	s	e		M	A	T	X	X	X		
i	n		t	h	a	t		X	X	X	.

```
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.

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

% Traverse the matrix to censor string str
```

zeros returns an array of type **double**

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

% 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
```

Matrix vs. Cell Array

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

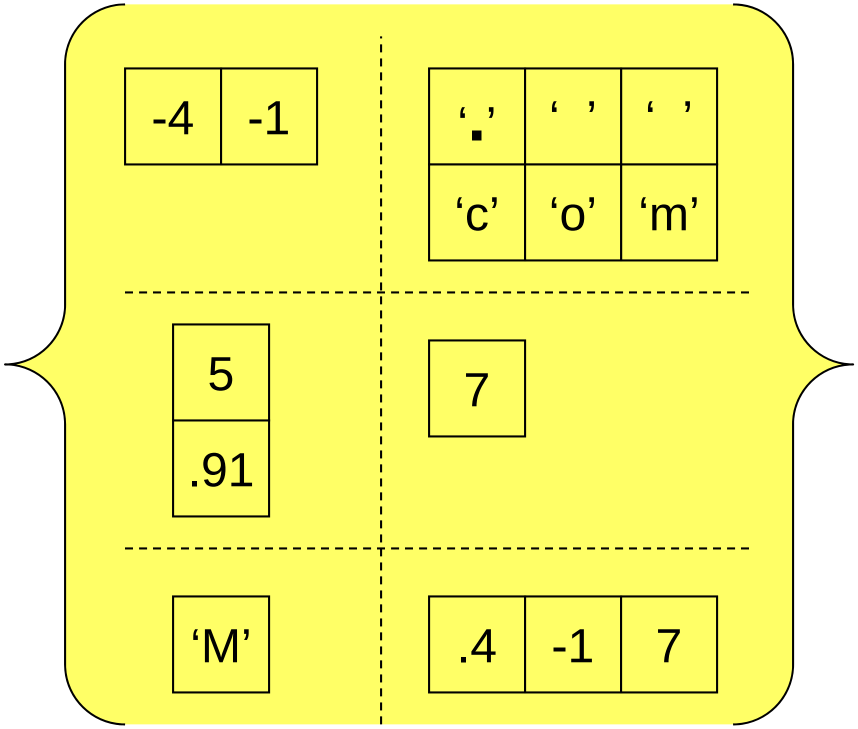
A cell array is a special array whose individual components may contain different types of data

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’

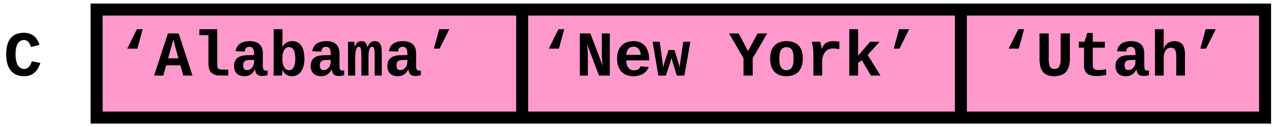
4 x 5 matrix



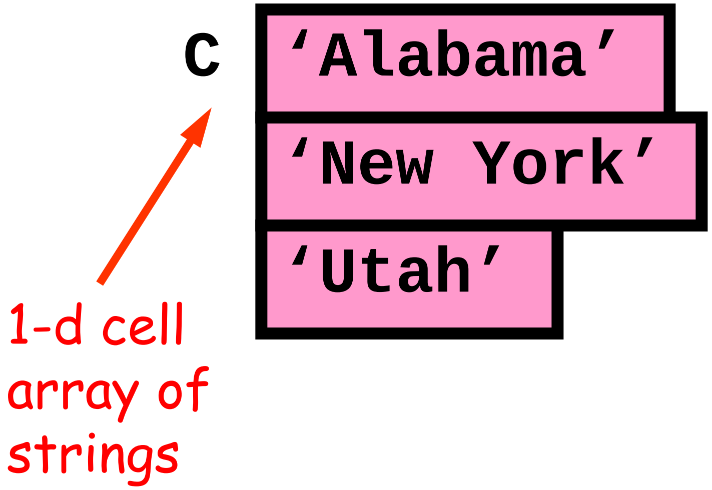
3 × 2 cell array

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    ’]
```

M	‘A’	‘l’	‘a’	‘b’	‘a’	‘m’	‘a’	‘ ‘
	‘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

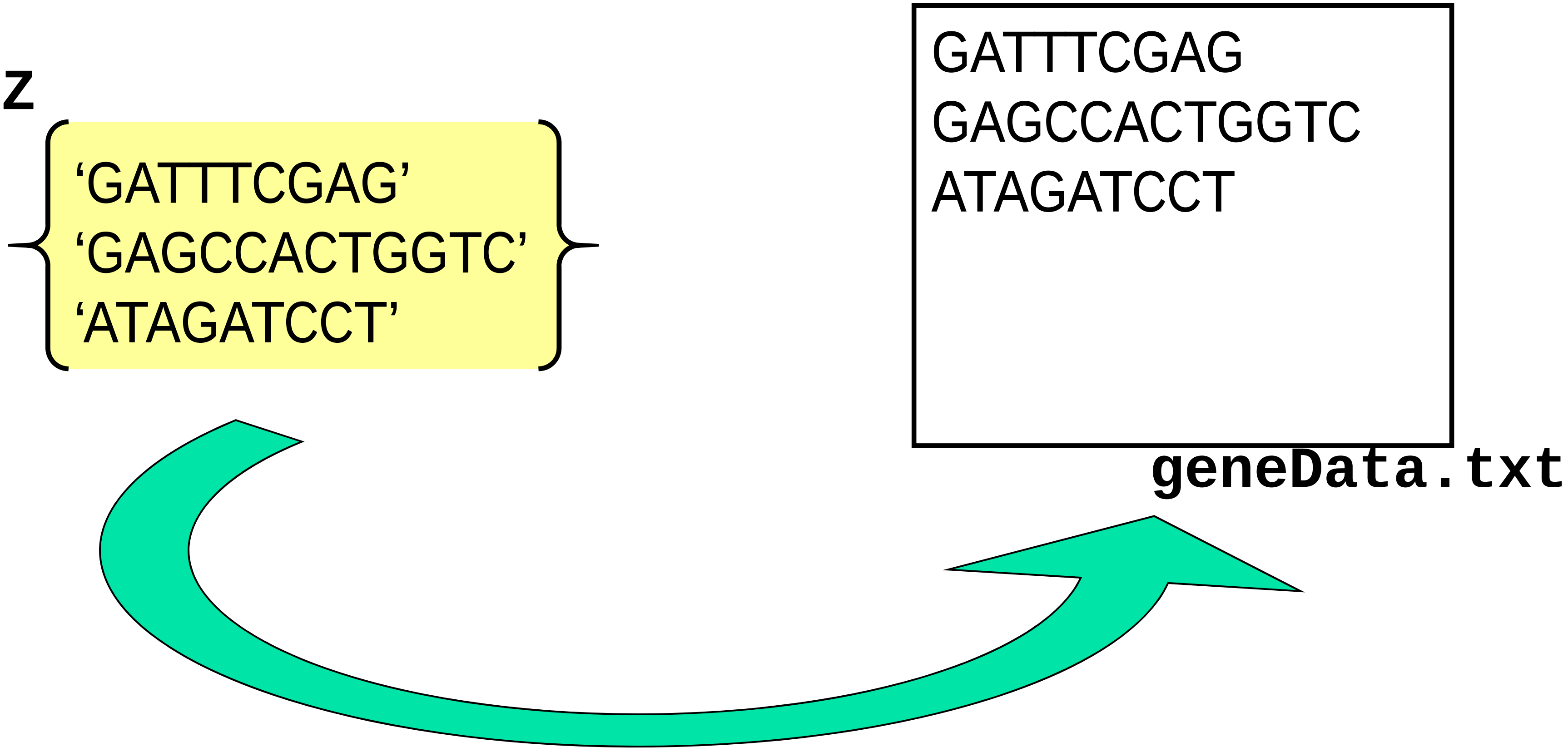
```
i = 1;  % index of next card

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
```

See function **CardDeck**

I want to put in the 3rd cell of cell array C a single string. 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

Example: Write a cell array of gene sequences to a file



z

‘GATTTCGAG’
‘GAGCCACTGGTC’
‘ATAGATCCT’

GATTTCGAG
GAGCCACTGGTC
ATAGATCCT

geneData.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

```
fid = fopen(‘geneData.txt’, ‘w’);
```

An open 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

2. Write (print) to the file

```
fid = fopen(‘geneData.txt’, 'w');

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

2. Write (print) to the file

```
fid = fopen(‘geneData.txt’, 'w');

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

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

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

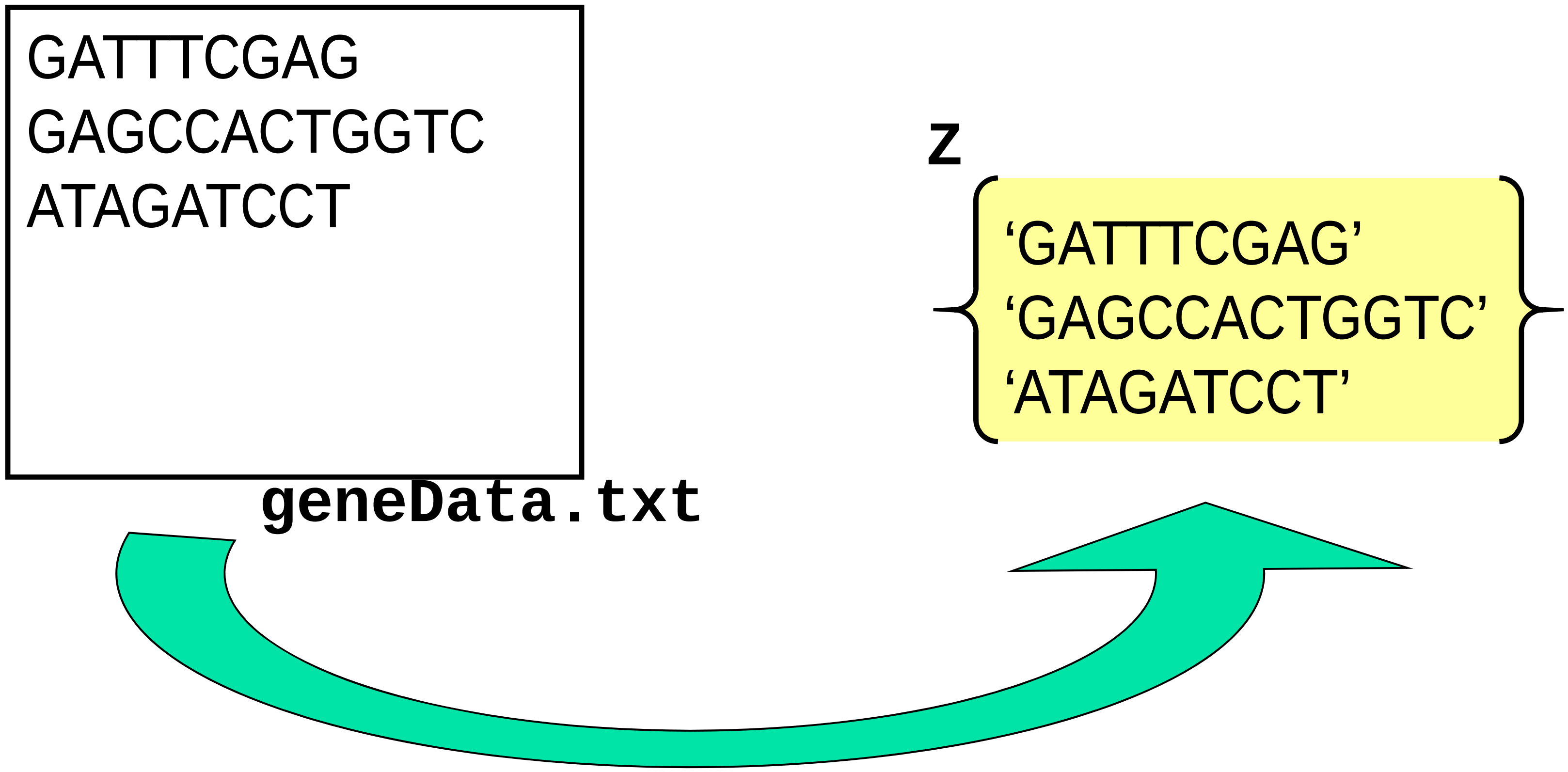
The **i**th item in cell array **Z**

3. Close the file

```
fid = fopen(‘geneData.txt’ ,'w');

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

fclose(fid);
```

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

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

Reverse problem: Read the data in a file line-by-line and store the results in a cell array



GATTTCGAG
GAGCCACTGGTC
ATAGATCCT

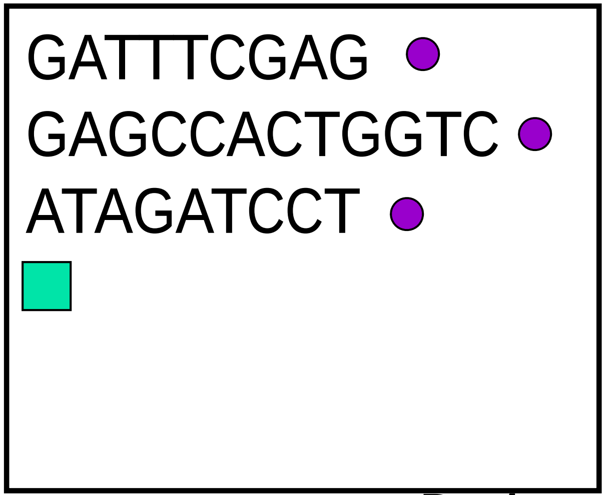
`geneData.txt`

`Z`

{ 'GATTTCGAG'
'GAGCCACTGGTC'
'ATAGATCCT' }

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

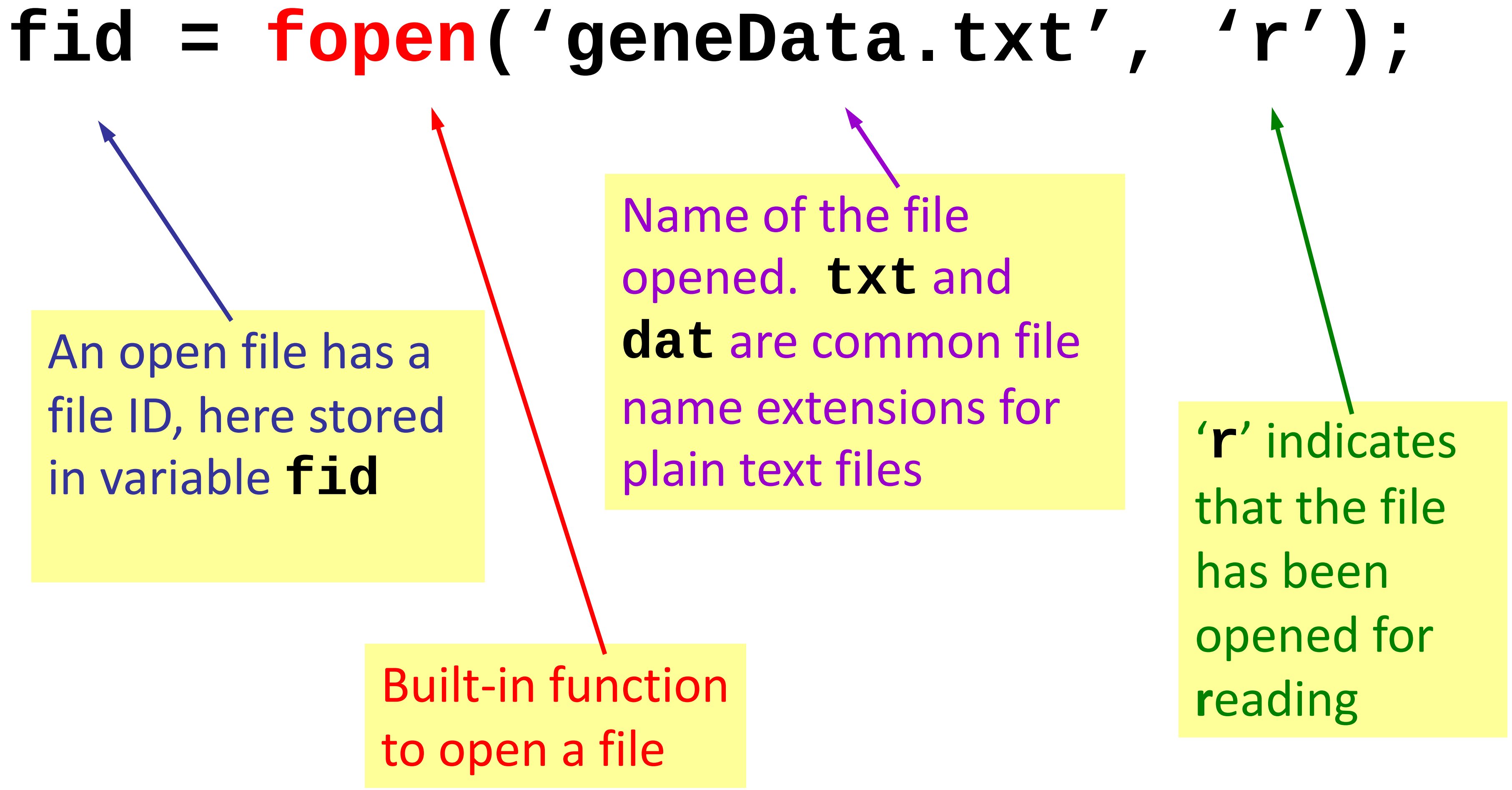
● Carriage return marks the end of a line

🟩 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

1. 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)
   k= k+1;
   Z{k}= fgetl(fid);
end
```

False until end-of-file is reached

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);
```

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

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