Announcements

- A2 resubmits due Wed
- Exams returned next week
  - Optional retest TBD
- A3 to be released next week, due Mon, Oct. 18

Su	M	Tu	W	Th	F	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Agenda

- Heterogeneous, nestable arrays
- Reading and writing files

Limitations of primitive arrays

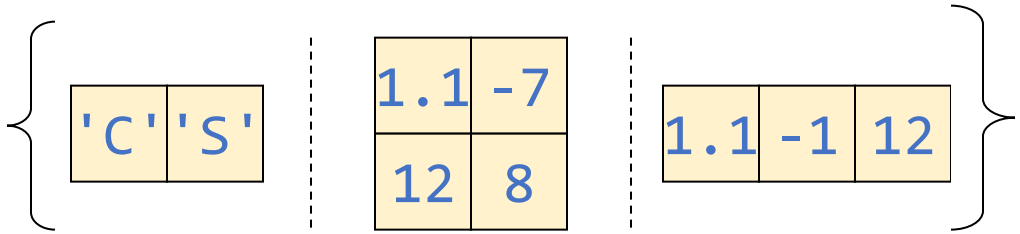
- Homogeneous data type
  - Can't represent tables
- Not nestable
  - No ragged arrays, lists-of-lists
  - Concatenation always "flattens"
- Poor support for strings

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

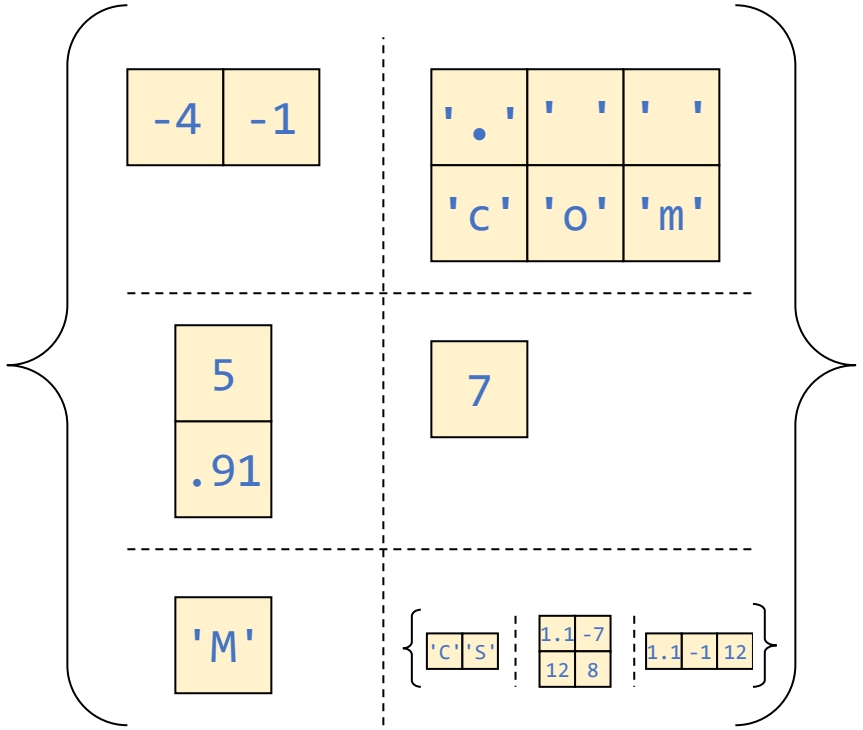
- `['John Doe', 33, true]`
  - Error using **horzcat**
- `[1, 2, 3; ... 4, 5]`
  - Error: Invalid expression.
- `[1, [2, 3], 4]`
  - 1 2 3 4

New data type: Cell

- A cell's value may be of any type
  - Array of doubles
  - Array of characters
  - Array of more cells
- Each cell in an array may have a different type & size



- Arrays of cells are still rectangular



Application: lists of strings

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

'Alabama'	'New York'	'Utah'
1	2	3

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

1	'Alabama'
2	'New York'
3	'Utah'

Compare with:

1,:	'A'	'l'	'a'	'b'	'a'	'm'	'a'	' '
2,:	'N'	'e'	'w'	' '	'Y'	'o'	'r'	'k'
3,:	'U'	't'	'a'	'h'	' '	' '	' '	' '

Use braces for creating & indexing cell arrays

Primitive arrays

- Create

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

- Index

```
m(2,1) = pi
disp(m(3,2))
```

Cell arrays

- Create

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

- Index

```
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= cells(1,3);  % optional
C{1}= 'Oct';
C{2}= 30;
C{3}= ones(3,2);
```

Can assign empty cell array

```
D= {};
```

Comparison of bracket operators

- Square brackets [ ]
  - Create primitive array
  - Concatenate (any) array contents

```
[3[14]1[59]]
[ 'a' {'b' ['c' 'd']} ] ⇒
{ 'a', 'b', 'cd' }
```

- Curly braces { }
  - Create cell array enclosing contents

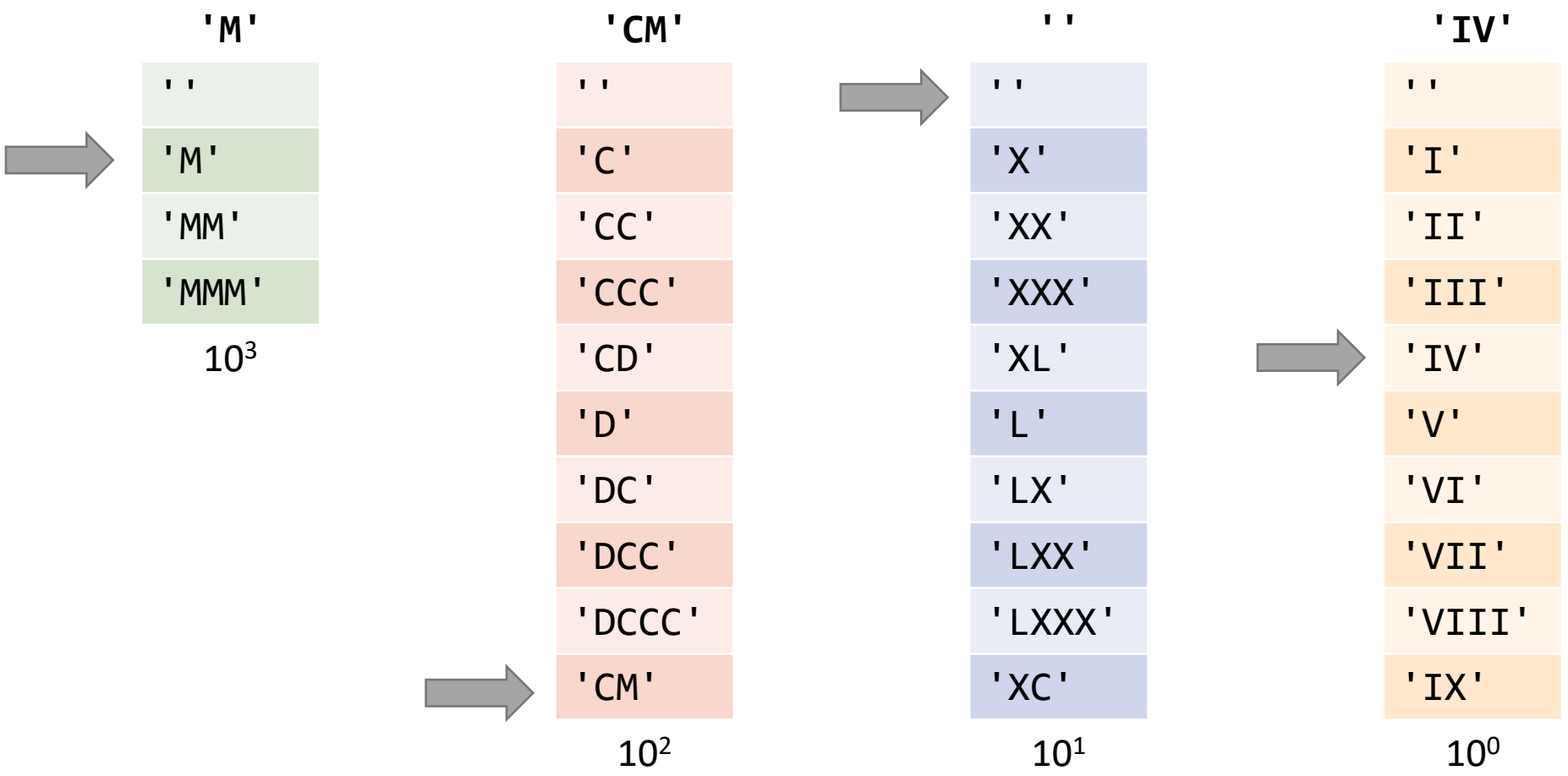
```
{ 3   [1 4]  1  [5 9] }
{ 'a' {'b' 'cd'} }
```

Example: Roman numerals

Goal: make a list of Roman numerals from 1-3999

```
C{1} = 'I'
C{2} = 'II'
C{3} = 'III'
   ⋮
C{2007} = 'MMVII'
   ⋮
C{3999} = 'MMMCMXCIX'
```


1
9
0
4


'M'
''
'M'
'MM'
'MMM'
10^3
'CM'
''
'C'
'CC'
'CCC'
'CD'
'D'
'DC'
'DCC'
'DCCC'
'CM'
10^2
''
''
'X'
'XX'
'XXX'
'XL'
'L'
'LX'
'LXX'
'LXXX'
'XC'
10^1
'IV'
''
'I'
'II'
'III'
'IV'
'V'
'VI'
'VII'
'VIII'
'IX'
10^0

Indexing quiz

```
digits = { {'','I',…,'IX'}; ...
           {'','X',…,'XC'}; ...
           {'','C',…,'CM'}; ...
           {'','M','MM','MMM'} };
```

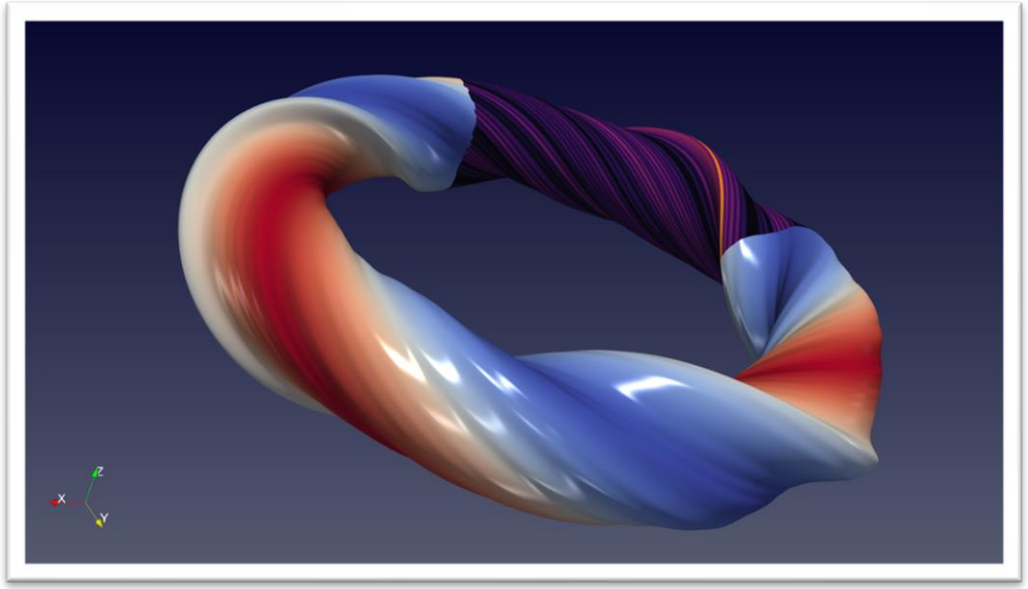
What letters represent 20 (= 2×10^1) ?

Alpha	digits{2,1}
Bravo	digits{1}{2}
Charlie	digits{2}{3}
Delta	digits{1,3}

```
for d3 = 0:3
    for d2 = 0:9
        for d1 = 0:9
            for d0 = 0:9
                n = d3*10^3 + d2*10^2 + ...
                    d1*10^1 + d0*10^0;
                C{n} = ⟨ digits{4}{d3+1} digits{3}{d2+1} ...
                         digits{2}{d1+1} digits{1}{d0+1} ⟩;
            end
        end
    end
end
```

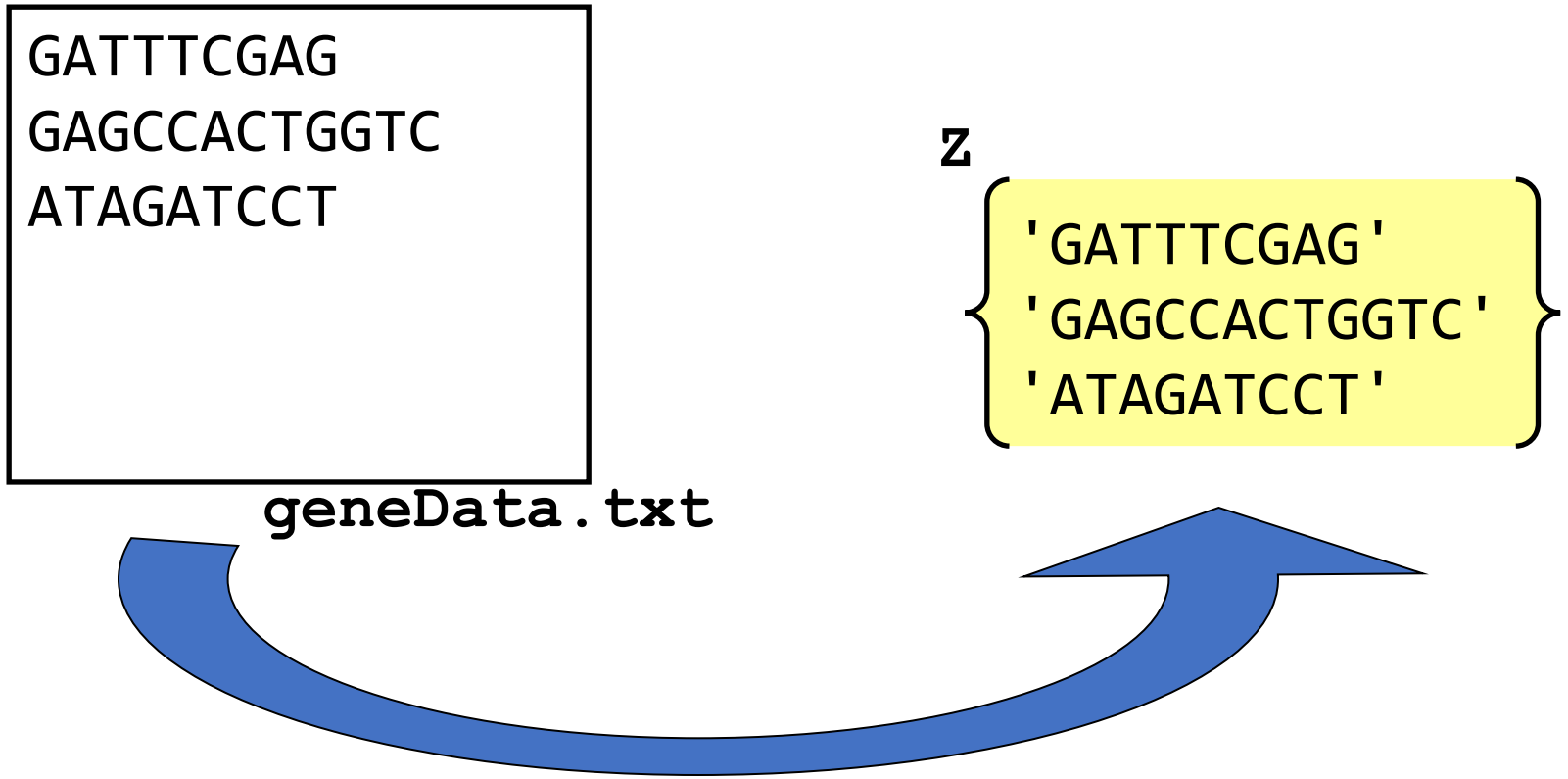
Reading and writing files

- Why?
  - Process data from the real world
  - Move data between programs best suited to each task



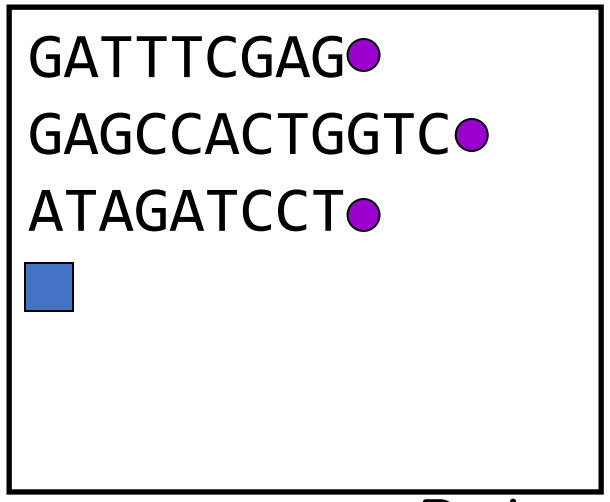
- 3-step process
  1. Open file (create or truncate if necessary)
  2. Read from or write to file
  3. Close file

Read lines into cell array



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

End-of-line and end-of-file



geneData.txt

- Carriage return and/or line feed characters mark the end of a line (`'\r\n'`)

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

  eof stands for end of file

Read lines into cell array

1. Open file
   - `fopen()`
2. Read it line-by-line until end-of-file
   - `fgetl()`, `feof()`
3. Close file
   - `fclose()`

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