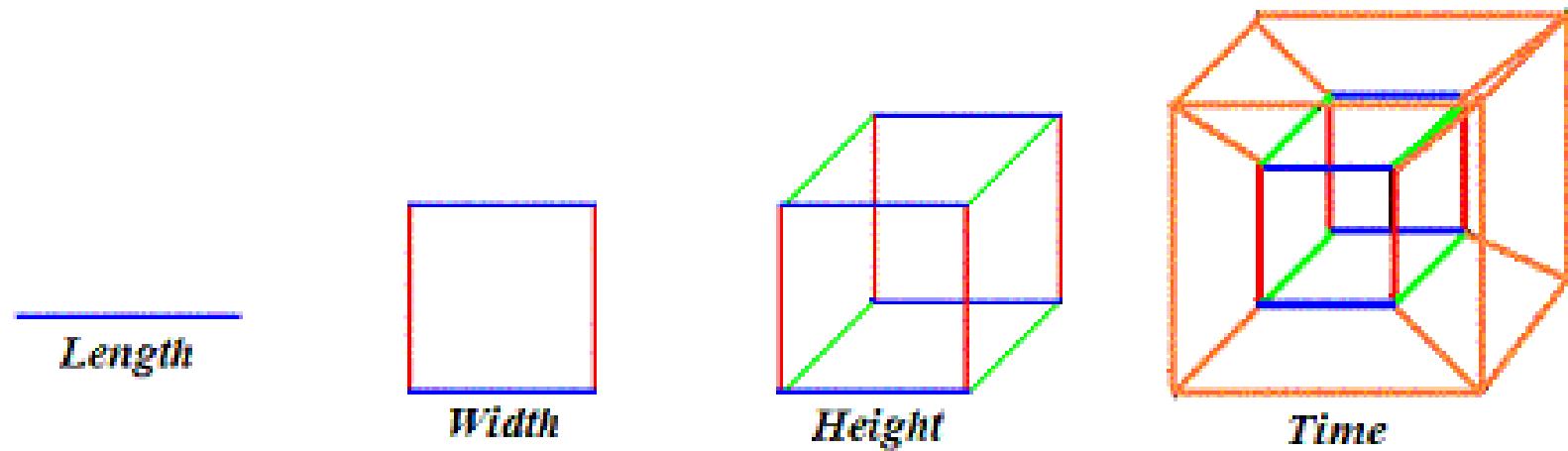


Dimensions

- Are we living in a three dimensional space?

The Four Dimensions



[The xkcd guide to the fourth dimension](#)

Two-dimensional Array

Motel in US (1D)

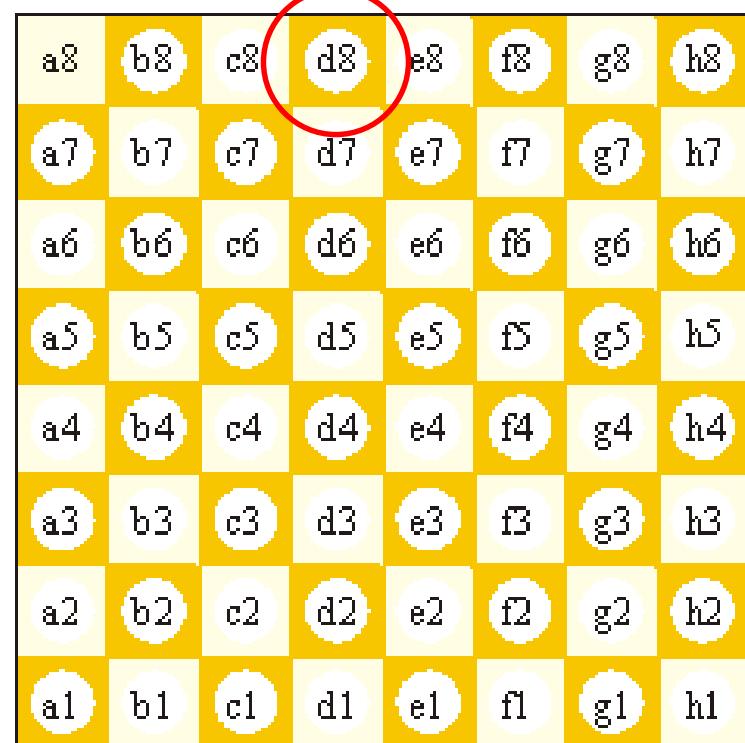
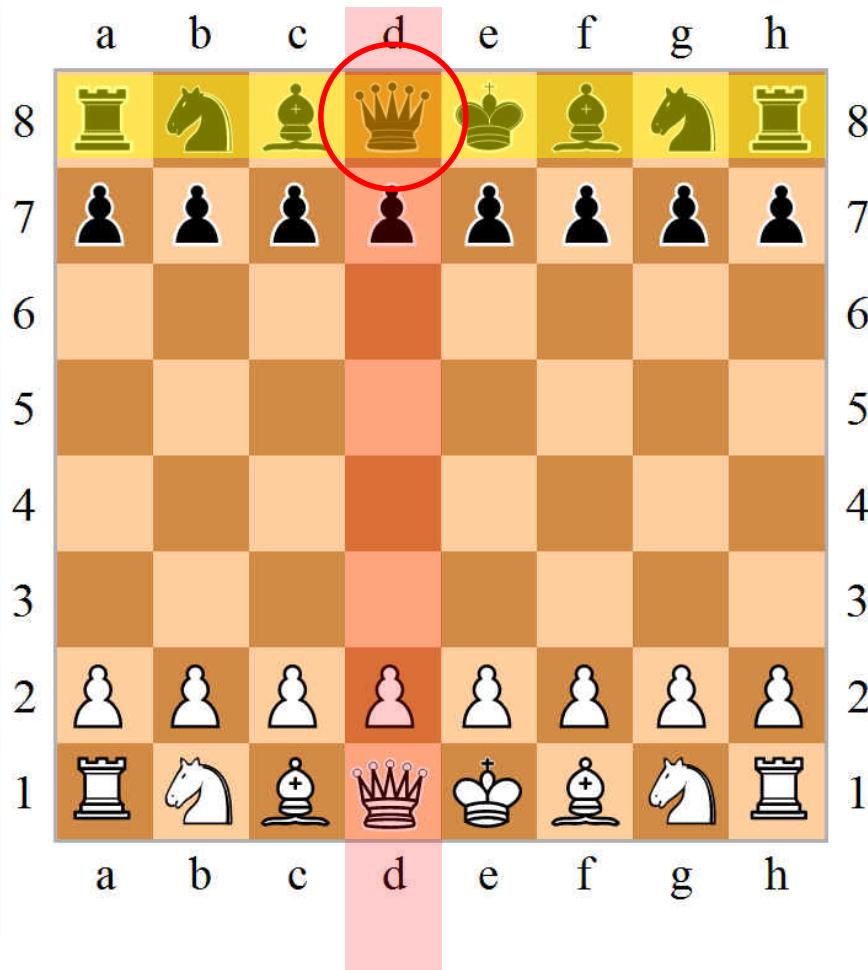


Hotel (2D)

My room is 02-05



Chess Position



The screenshot shows a Microsoft Excel spreadsheet titled "crude-...". The ribbon menu is visible at the top, with tabs for Home, Insert, Page Layout, Formulas, Data, Review, View, and a smiley face icon. The "View" tab is selected. The status bar at the bottom shows "Ready" and "150%".

The table in the spreadsheet has the following structure:

	A	B	C	D
1	year	level_1	value	
2	1960	Crude Birth Rate	37.5	
3	1961	Crude Birth Rate	35.2	
4	1962	Crude Birth Rate	33.7	
5	1963	Crude Birth Rate	33.2	
6	1964	Crude Birth Rate	31.6	
7	1965	Crude Birth Rate	29.5	
8	1966	Crude Birth Rate	28.3	
9	1967	Crude Birth Rate	25.6	
10	1968	Crude Birth Rate	23.5	
11	1969	Crude Birth Rate	21.8	
12	1970	Crude Birth Rate	22.1	
13	1971	Crude Birth Rate	22.3	
14	1972	Crude Birth Rate	23.1	

The cell B8 is highlighted in yellow, and the entire column B is highlighted in orange. The formula bar at the top shows "Crude Birth Rate".

- Row 8
- Col B
- So, B8

Dimensions

One Dimensional Array, A

Index	Contents
0	'Apple'
1	'John'
2	'Eve'
3	'Mary'
4	'Ian'
5	'Smith'
6	'Kelvin'

A[5] = 'Smith'

Two Dimensional Array, M

	0	1	2	3
0	'Apple'	'Lah'	'Cat'	'Eve'
1	'Hello'	'Pay'	'TV'	'Carl'
2	'What'	'Bank'	'Radio'	'Ada'
3	'Frog'	'Peter'	'Sea'	'Eat'
4	'Job'	'Fry'	'Gym'	'Wow'
5	'Walk'	'Fly'	'Cook'	'Look'

M[4][1] = 'Fry'

2-Dimensional Array in Python

- 1-d array

```
>>> l = [1,2,3,4,5]  
>>> l[3]  
4
```

- 2-d array

```
>>> m = [[1,2], [3,4]]  
>>> m[0][0]  
1  
>>> m[0][1]  
2  
>>> m[1][0]  
3  
>>> m[1][1]  
4
```

Column index

Row index

- Equivalent to

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

What is the Difference Between ...

- A list of lists
 - More specifically, all the lists do not have to be the same length
- 2D Array
 - Is a subset of a list of lists
 - #rows, Can be any length
 - Length = 3
 - [[1,2],[3,4,5],[6]]
 - Length = 2
 - Length = 3
 - Length = 1
 - Length = 3
 - [[1,2],[3,4],[5,6]]
 - Length = 2
 - Length = 2
 - Length = 2
 - ↔
 - $$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$
 - Can be any length, but **all** the lists have to be the **same** length

A Larger 2-d Array

- Creating a 4×10 matrix

```
>>> m2 = [[11,12,13,14,15,16,17,18,19,20], [21,22,23,24,25,26,27,28,29,30],  
           [31,32,33,34,35,36,37,38,39,40], [41,42,43,44,45,46,47,48,49,50]]  
>>> m2  
[[11, 12, 13, 14, 15, 16, 17, 18, 19, 20], [21, 22, 23, 24, 25, 26, 27, 28,  
     29, 30], [31, 32, 33, 34, 35, 36, 37, 38, 39, 40], [41, 42, 43, 44, 45,  
     46, 47, 48, 49, 50]]  
>>> pprint(m2) ← pprint() from  
          the package  
          pprint makes  
          the format nicer  
[[11, 12, 13, 14, 15, 16, 17, 18, 19, 20],  
 [21, 22, 23, 24, 25, 26, 27, 28, 29, 30],  
 [31, 32, 33, 34, 35, 36, 37, 38, 39, 40],  
 [41, 42, 43, 44, 45, 46, 47, 48, 49, 50]]  
>>> len(m2) ← Number of rows  
4  
>>> len(m2[0]) ← Number of columns  
10
```

To Create an $N \times N$ Identical Matrix

```
def identityMatrix(N):
    output = []
    for i in range(N):
        row = []
        for j in range(N):
            row.append(1 if i == j else 0)
        output.append(row)
    return output

pprint(identityMatrix(10))
```

```
[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],  
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]
```

Row i _____

Column j _____

2D Looping

- Let's say we just want to print 0 and 1

```
[[1, 0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 1, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 1, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 1, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 1, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 1, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 1, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 1, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0, 1]]
```



```
1000000000  
0100000000  
0010000000  
0001000000  
0000100000  
0000010000  
0000001000  
0000000100  
0000000010  
0000000001
```

2D Looping

- Let's say we just want to print 0 and 1

```
def mTightPrint(m) :  
    for i in range(len(m)) :  
        line = ''  
        for j in range(len(m[0])) :  
            line += str(m[i][j])  
        print(line)  
  
mTightPrint(m1)
```

↑
Row i
↑
Column j

j = 0 1 2 3 4 5 6 7 8 9	↓↓↓↓↓↓↓↓↓↓
i = 0 →	1000000000
i = 1 →	0100000000
i = 2 →	0010000000
i = 3 →	0001000000
i = 4 →	0000100000
i = 5 →	0000010000
i = 6 →	0000001000
i = 7 →	0000000100
i = 8 →	0000000010
i = 9 →	0000000001

To Create an R x C Zero Matrix

```
def createZeroMatrix(r,c):  
    output = []  
    for i in range(r):  
        row = []  
        for j in range(c):  
            row.append(0)  
        output.append(row)  
    return output
```

```
>>> m = createZeroMatrix(4,9)  
>>> m  
[[0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0]]  
>>> pprint(m)  
[[0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0]]
```

Accessing Entries

- Remember
 - First row index
 - Then column index

i = 1 →

```
>>> pprint(m)
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
>>> m[1][3] = 9
>>> m[3][7] = 6
>>> pprint(m)
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 9, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 6, 0, 0]]
```

↑
j = 3

2D Array Applications

- Matrix operations
- Recoding 2D data
 - Tables
 - Spatial Data

Matrix Addition

```
def sumMatrices(m1,m2): # Assuming m1 and m2 have the same #r,c
    r = len(m1)
    c = len(m1[0])
    output = createZeroMatrix(r,c)
    for i in range(r):
        for j in range(c):
            output[i][j] = m1[i][j] + m2[i][j]
    return output
```

- How do to Matrix Multiplication or other matrix operations, e.g. inverse, transpose?

```
>>> pprint(m1)
[[4, 7, 6, 1, 1, 2, 2, 1, 8],
 [7, 0, 3, 5, 7, 3, 3, 9, 2],
 [6, 3, 1, 3, 2, 2, 5, 8, 9],
 [4, 0, 3, 2, 9, 3, 6, 8, 0]]
>>> pprint(m2)
[[2, 0, 2, 5, 3, 9, 3, 2, 4],
 [0, 4, 7, 2, 6, 5, 1, 7, 5],
 [6, 0, 6, 7, 5, 9, 0, 7, 7],
 [0, 8, 1, 5, 2, 5, 9, 1, 3]]
>>> pprint(sumMatrices(m1,m2))
[[6, 7, 8, 6, 4, 11, 5, 3, 12],
 [7, 4, 10, 7, 13, 8, 4, 16, 7],
 [12, 3, 7, 10, 7, 11, 5, 15, 16],
 [4, 8, 4, 7, 11, 8, 15, 9, 3]]
```

Tables

- Just like other Spreadsheet applications

Name	Stu. No.	English	Math	Science	Social Studies
John	A1000000A	90	80	100	70
Peter	A1000009D	60	100	60	90
Paul	A1000003C	80	80	70	90
Mary	A1000001B	100	70	80	80

```
>>> records = [['John', 'A1000000A', 90, 80, 100, 70],  
                 ['Peter', 'A1000009D', 60, 100, 60, 90],  
                 ['Paul', 'A1000003C', 80, 80, 70, 90],  
                 ['Mary', 'A1000001B', 100, 70, 80, 80]]  
  
>>> records[2][5] = 100  
  
>>> pprint(records)  
[['John', 'A1000000A', 90, 80, 100, 70],  
 ['Peter', 'A1000009D', 60, 100, 60, 90],  
 ['Paul', 'A1000003C', 80, 80, 70, 100],  
 ['Mary', 'A1000001B', 100, 70, 80, 80]]
```

Tables

- Like in Spreadsheet applications

Name	Stu. No.	English	Math	Science	Social Studies
John	A1000000A	90	80	100	70
Peter	A1000009D	60	100	60	90
Paul	A1000003C	80	80	70	90
Mary	A1000001B	100	70	80	80

- Or, you can setup a more “comprehensible” column index dictionary

```
>>> colIndex = {'name':0, 'SN':1, 'eng':2, 'math':3, 'sci':4, 'sos':5}  
>>> records[3][colIndex['eng']] = 0  
>>> pprint(records)  
[['John', 'A1000000A', 90, 80, 100, 70],  
 ['Peter', 'A1000009D', 60, 100, 60, 90],  
 ['Paul', 'A1000003C', 80, 80, 70, 100],  
 ['Mary', 'A1000001B', 0, 70, 80, 80]]
```

Tables

- You can even append a new record for a new student

```
>>> records.append(['Seon', 'A1000004Z', 70, 80, 70, 80])
>>> pprint(records)
[['John', 'A1000000A', 90, 80, 100, 70],
 ['Peter', 'A1000009D', 60, 100, 60, 90],
 ['Paul', 'A1000003C', 80, 80, 70, 100],
 ['Mary', 'A1000001B', 0, 70, 80, 80],
 ['Seon', 'A1000004Z', 70, 80, 70, 80]]
```

Tables

- Or do any computations on the table
 - E.g. The average of the English scores

```
>>> pprint(records)
[['John', 'A1000000A', 90, 80, 100, 70],
 ['Peter', 'A1000009D', 60, 100, 60, 90],
 ['Paul', 'A1000003C', 80, 80, 70, 100],
 ['Mary', 'A1000001B', 0, 70, 80, 80],
 ['Seon', 'A1000004Z', 70, 80, 70, 80]]
>>> scoreSumEng = 0
>>> for i in range(len(records)):
    scoreSumEng += records[i][colIndex['eng']]
>>> print(scoreSumEng/len(records))
60.0
```

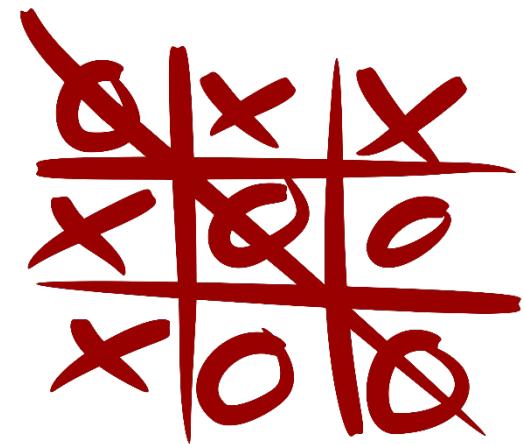
Spatial Data

Tic-tac-toe, Chess, SRPG

Tic-tac-toe

- Tic-tac-toe
 - A game of 3×3 grid
 - How to represent the game?
 - A 3×3 Matrix:

```
game = createZeroMatrix(3,3)
```
- How to "make a move"?
 - You can directly ask the user to input a pair of coordinates, e.g. (1,1)
 - However, sometime it's confusing for users who cannot tell row indices from column indices



Tic-tac-toe

- Because the grid size is very small, we can make it easier for the user by labeling each box with a number from 1 to 9
- For the position pos from 1 to 9, we can map to the coordinates (i, j) as

$$i = (pos-1)//3$$

$$j = (pos-1)\%3$$

1|2|3

4|5|6

7|8|9

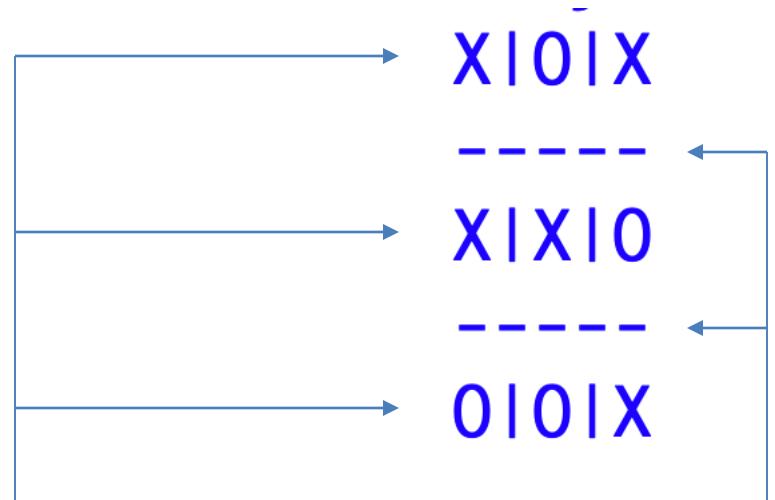
- Challenge

- Given the row and column indices i and j, how to compute the position pos?

A Simple TTT Game

- First, for a matrix **game** that represent the current state of the game, let's make a function to print it nicely

```
def printTTT(game):  
    for i in range(3):  
        print(f'{game[i][0]}|{game[i][1]}|{game[i][2]}')  
        if i !=2:  
            print( '-----')
```



A Simple TTT Game

```
def tttGamePlay():
    game = createZeroMatrix(3,3)
    for i in range(3):
        for j in range(3):
            game[i][j] = i*3+j+1
    player = 0
    piece = ['X', 'O']
    printTTT(game)
    for i in range(9): # Anyhow play 9 times
        pos = int(input(f'Player {piece[player]} move:')) - 1
        game[pos//3][pos%3] = piece[player]
        player = 1 - player
    printTTT(game)
```

Create the game board

A Simple TTT Game

```
def tttGamePlay():
    game = createZeroMatrix(3,3)
    for i in range(3):
        for j in range(3):
            game[i][j] = i*3+j+1
    player = 0
    piece = ['X','0']
    printTTT(game)
    for i in range(9): # Anyhow play 9 times
        pos = int(input(f'Player {piece[player]} move:')) - 1
        game[pos//3][pos%3] = piece[player]
        player = 1 - player
    printTTT(game)
```

Put the number 1 to 9 into the board for display purposes

1|2|3

4|5|6

7|8|9

A Simple TTT Game

```
def tttGamePlay():
    game = createZeroMatrix(3,3)
    for i in range(3):
        for j in range(3):
            game[i][j] = i*3+j+1
    player = 0
    piece = ['X','O']
    printTTT(game)
    for i in range(9): # Anyhow play 9 times
        pos = int(input(f'Player {piece[player]} move:')) - 1
        game[pos//3][pos%3] = piece[player]
        player = 1 - player
    printTTT(game)
```

There are two players,
Player 0 and Player 1
Player 0 uses 'X'
Player 1 uses 'O'

The first player is Player 0

And they will take turns

A Simple TTT Game

```
def tttGamePlay():
    game = createZeroMatrix(3,3)
    for i in range(3):
        for j in range(3):
            game[i][j] = i*3+j+1
    player = 0
    piece = ['X', 'O']
    printTTT(game)
    for i in range(9): # Anyhow play 9 times
        pos = int(input(f'Player {piece[player]} move:')) - 1
        game[pos//3][pos%3] = piece[player]
        player = 1 - player
    printTTT(game)
```

Anyhow play 9 times
(We didn't implement any error or winning checking yet)

Gameplay

1|2|3

4|5|6

7|8|9

Player 0 move:3

1|2|0

4|X|6

7|8|9

Player X move:5

1|2|3

4|X|6

7|8|9

Player X move:6

1|2|0

4|X|X

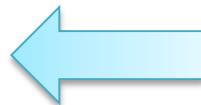
7|8|9

Chess

```
def createChessGame():
    game = createZeroMatrix(8, 8)
    for j in range(8):
        game[1][j] = 'p'
        game[6][j] = 'P'
    game[7] = ['R', 'K', 'B', 'Q', 'E', 'B', 'K', 'R']
    game[0] = list(map(lambda x:x.lower(), game[7]))
    return game
```

```
game = createChessGame()
mTightPrint(game)
```

```
rkbqebkr
pppppppp
00000000
00000000
00000000
00000000
PPPPPPPP
RKBQEBKR
```



Other Text Based Chess

	A	B	C	D	E	F	G	H
8	r	n	b	q	k	b	n	r
7	p	p	p	p	p	p	p	p
6
5
4
3
2	P	P	P	P	P	P	P	P
1	R	N	B	Q	K	B	N	R

	A	B	C	D	E	F	G	H
8	#	#	#	#	#	#	#	#
7	R	N	B	Q	K	B	N	R
6	#	#	#	#	#	#	#	#
5	P	P	P	P	P	P	P	P
4	#	#	#	#	#	#	#	#
3	#	#	#	#	#	#	#	#
2	#	#	#	#	#	#	#	#
1	#	P	#	P	#	P	#	P
0	R	N	B	Q	K	B	N	R

```
worldwindow@Blackice: /home/worldwindow/c++_projekte/cchess# ./cc
### ConChess - A console based chess game #####
Author: Christian Guckelsberger Version 0.01

---A-----B-----C-----D-----E-----F-----G-----H-----
1 |Castle1| Horse1 | Bishop1 | Queen  | King   | Bishop2| Horse2 | Castle2 |
2 | Pawn1 | Pawn2  | Pawn3  | Pawn4  | Pawn5 | Pawn6  | Pawn7 | Pawn8  |
3 |-----|-----|-----|-----|-----|-----|-----|-----|
4 |-----|-----|-----|-----|-----|-----|-----|-----|
5 |-----|-----|-----|-----|-----|-----|-----|-----|
6 |-----|-----|-----|-----|-----|-----|-----|-----|
7 | Pawn1 | Pawn2  | Pawn3  | Pawn4  | Pawn5 | Pawn6  | Pawn7 | Pawn8  |
8 |Castle1| Horse1 | Bishop1| King   | Queen  | Bishop2| Horse2 | Castle2 |

++Player1++
Please enter the name of the figure you want to move: Q
```

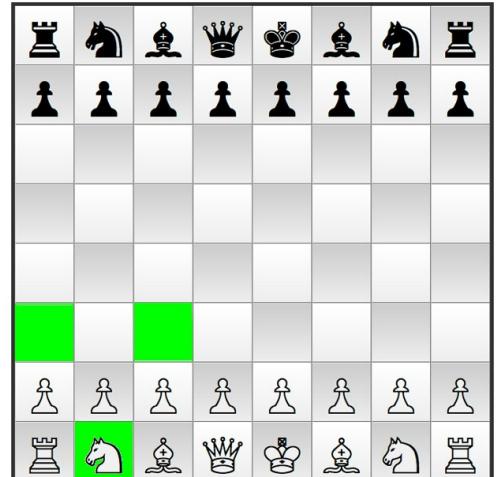
Chess

- How to make a move?
- E.g. moving a knight
 - Maybe

game[7][1] = 0

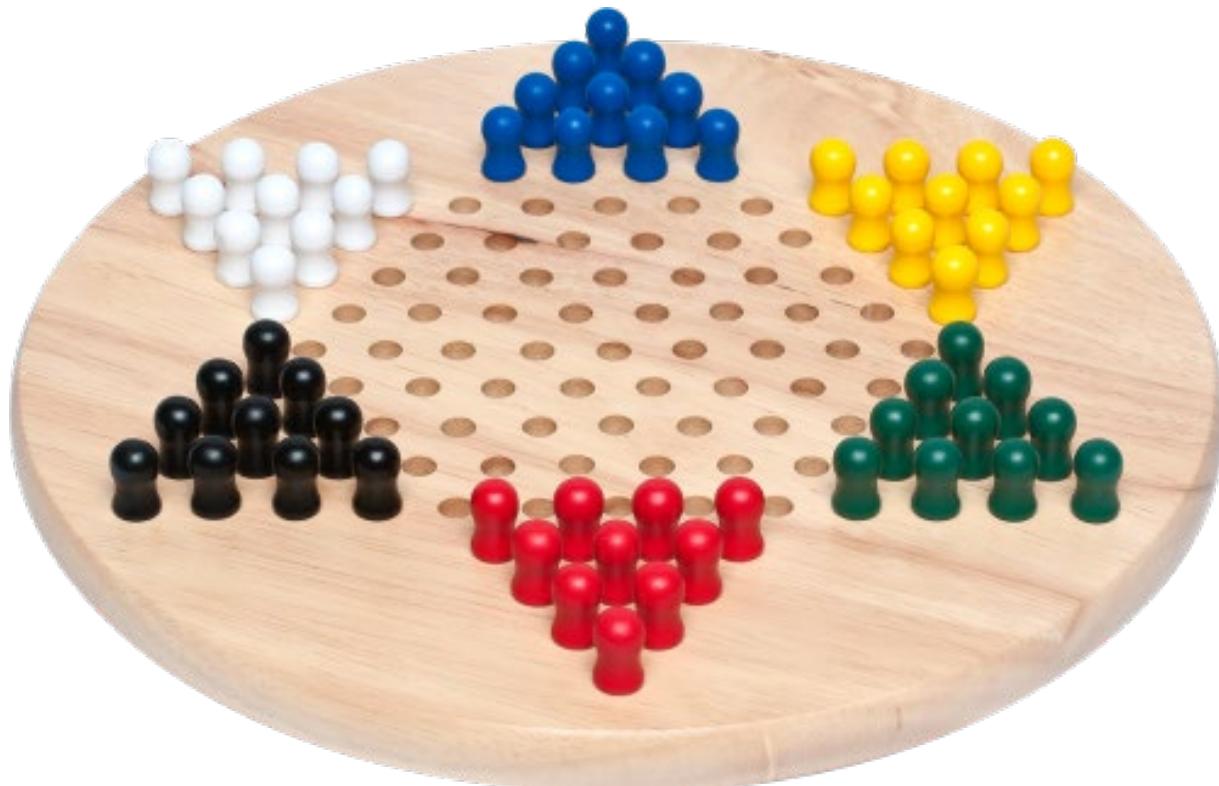
game[5][2] = 'K'

rkbqebkr
pppppppp
00000000
00000000
00000000
00000000
PPPPPPPP
RKBQEBKR



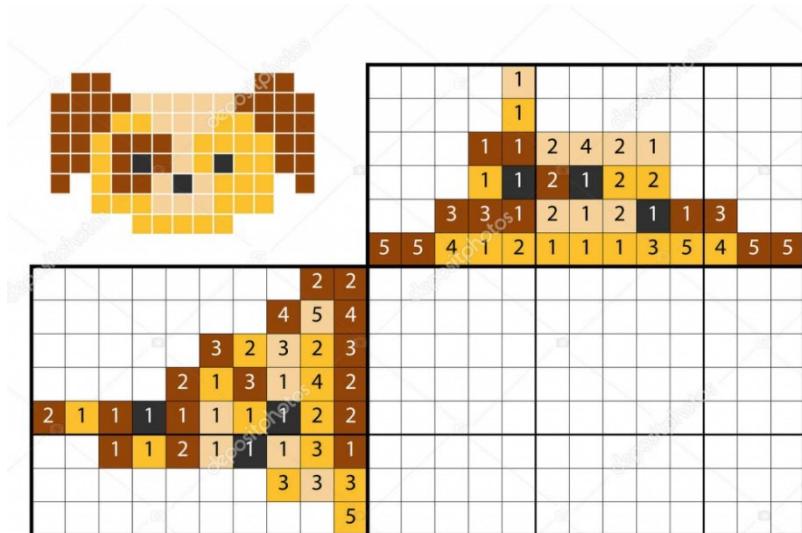
Chinese Checkers?

- How to represent the board?



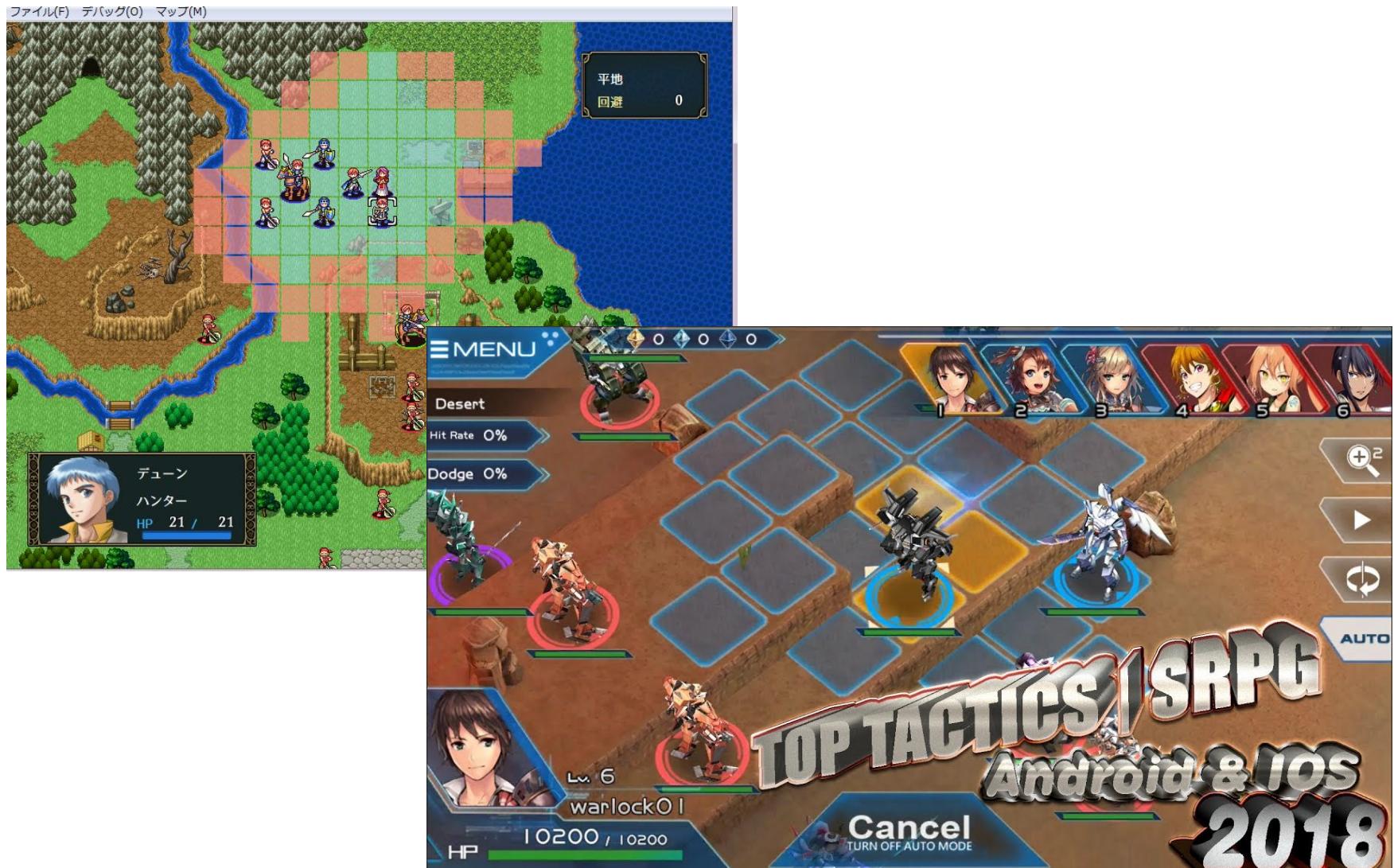
Other 2D Array Games

5	3		7			
6		1	9	5		
	9	8			6	
8			6			3
4		8	3			1
7		2			6	
6			2	8		
	4	1	9			5
		8		7	9	



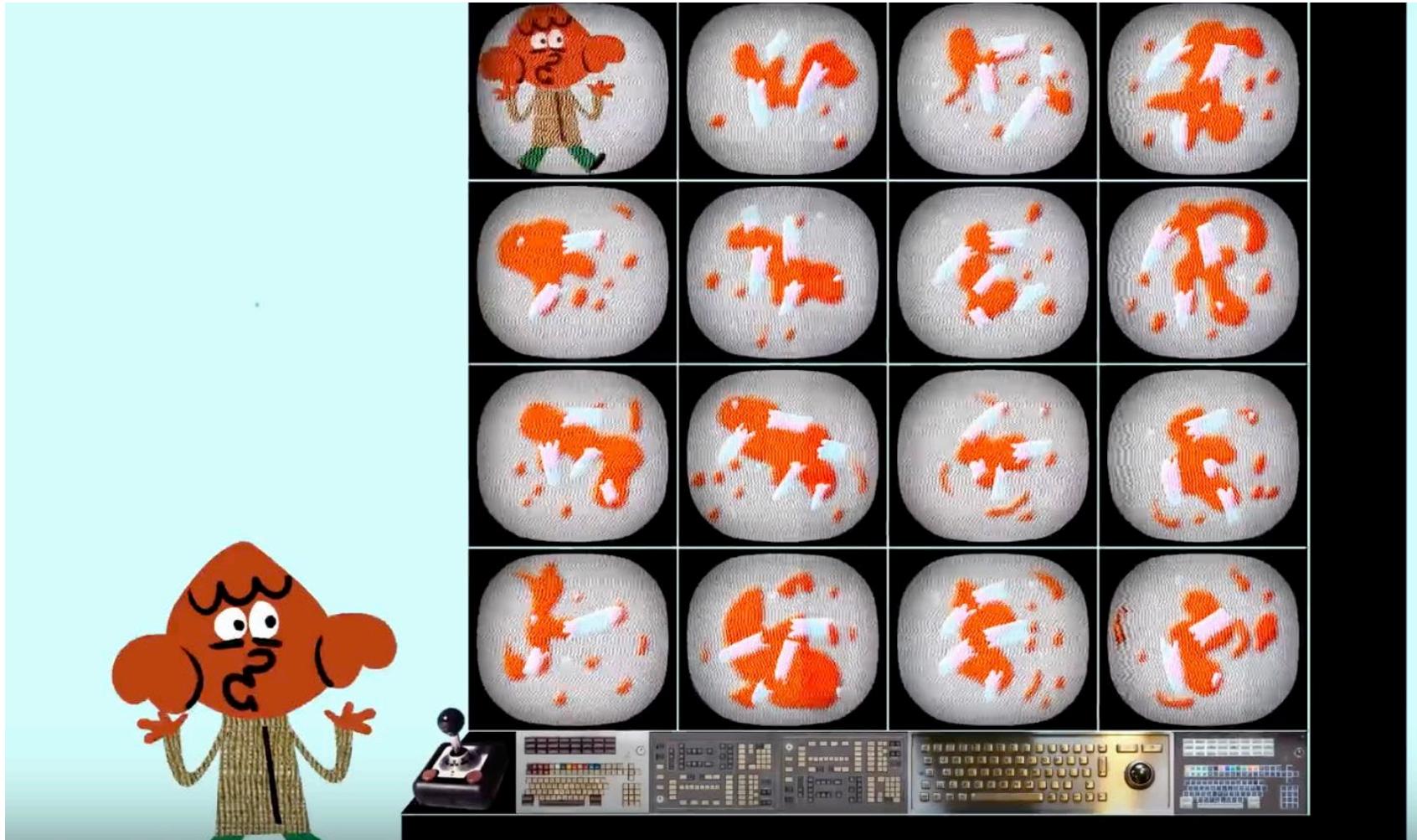
1		1	2	3	*	2	1	3	1	1	3	2	3
1			2	*	4	1	1	2	*	1	2		1
2	2	2	1	3	1	4	1	2	2	4	3	3	1
3	1	3	1	2	*	2		1	1	3	1	2	1
3	1	1	*	2	1	1	2	1	2	2	1	2	2
2	2	5	1	3	2	*	4	3	2	1		1	1
1	1	4	1	2	3	1	*	2	1	1	1	1	1
1	2	1	3	2	3	1	4	*	5	4	1	4	2
2	3	1	1	2	3	1	1	3	1	1	1	1	2
2	1	3	1	1	2	3	3	3	2	3	5	5	1

RPG, SRPG



Game of Life

- <https://bitstorm.org/gameoflife/>

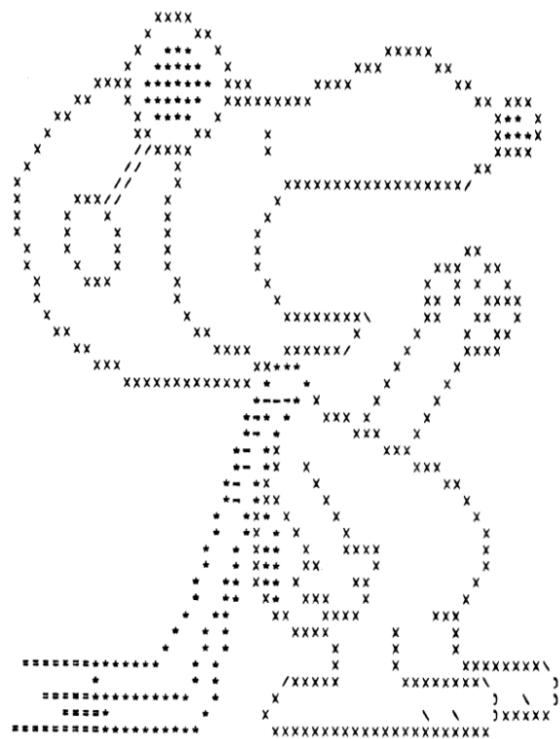


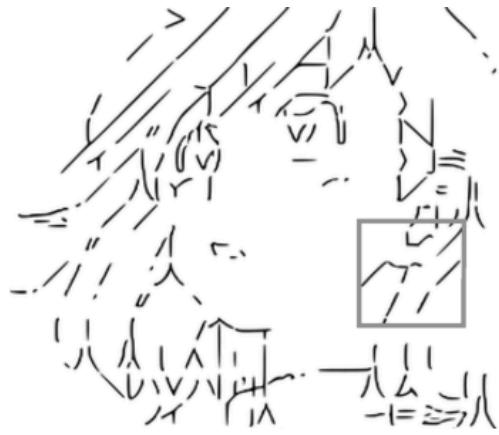
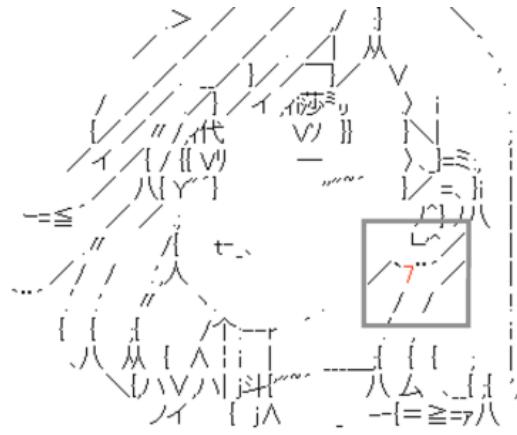
- Virus Riddle

circlePic()

```
def circlePic():
    l = 30
    center = 1/2
    pic = createZeroMatrix(l, l)
    for i in range(l):
        for j in range(l):
            if (l/3)**2 < (i-center)**2 + (j-center)**2 < (l/2)**2:
                pic[i][j] = "#"
            else:
                pic[i][j] = "."
    mTightPrint(pic)
```

ASCII Arts





Wait...

- What if I put three colors as each of the item in my 2D array?

```
[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]
```

[100,20
0,100]
Red, green, blue
= [100,200,100]

2D and 3D Arrays

2D Arrays

- A list of lists
- A vehicle loading with vehicles
 - And each vehicle contains some passengers



3D Arrays

- A list of lists of lists
- A vehicle loading with vehicles loading with vehicles
 - And each vehicle contains some passengers



$2+1 = 3D$ Array as a Picture

```
pic = [[[0,0,255], [255,0,0], [0,0,255]],  
        [[255,0,0], [255,0,0], [255,0,0]],  
        [[0,0,255], [255,0,0], [0,0,255]]]
```

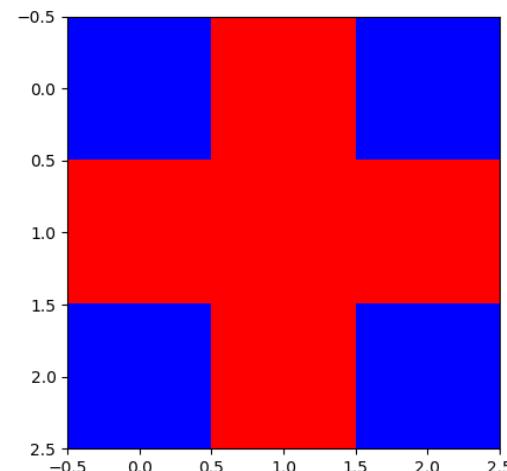
```
import matplotlib.pyplot as plt  
plt.imshow(pic)  
plt.show()
```

One pixel is a list of three values of red, blue and green.

Usually ranging from 0 to 255

`imshow()` is to draw a picture in bitmap

- 3×3 ($\times 3$ for colors) pixels (zoomed)



Recap: circlePic()

```
def circlePic():
    l = 30
    center = l/2
    pic = createZeroMatrix(l,l)
    for i in range(l):
        for j in range(l):
            if (l/3)**2 < (i-center)**2 + (j-center)**2 < (l/2)**2:
                pic[i][j] = "#"
            else:
                pic[i][j] = "."
    mTightPrint(pic)
```

The image shows a decorative border consisting of a repeating pattern of black dots and vertical lines. The pattern forms a diamond shape, with vertical lines on the left and right sides and horizontal lines connecting them. This pattern is repeated across the entire border.

```
def circlePic2():
    l = 300
    center = l/2
    pic = createZeroMatrix(l,l)
    for i in range(l):
        for j in range(l):
            if (l/3)**2 < (i-center)**2 + (j-center)**2 < (l/2)**2:
                pic[i][j] = [0,0,255]
            else:
                pic[i][j] = [255,0,0]
    plt.imshow(pic)
    plt.show()
```

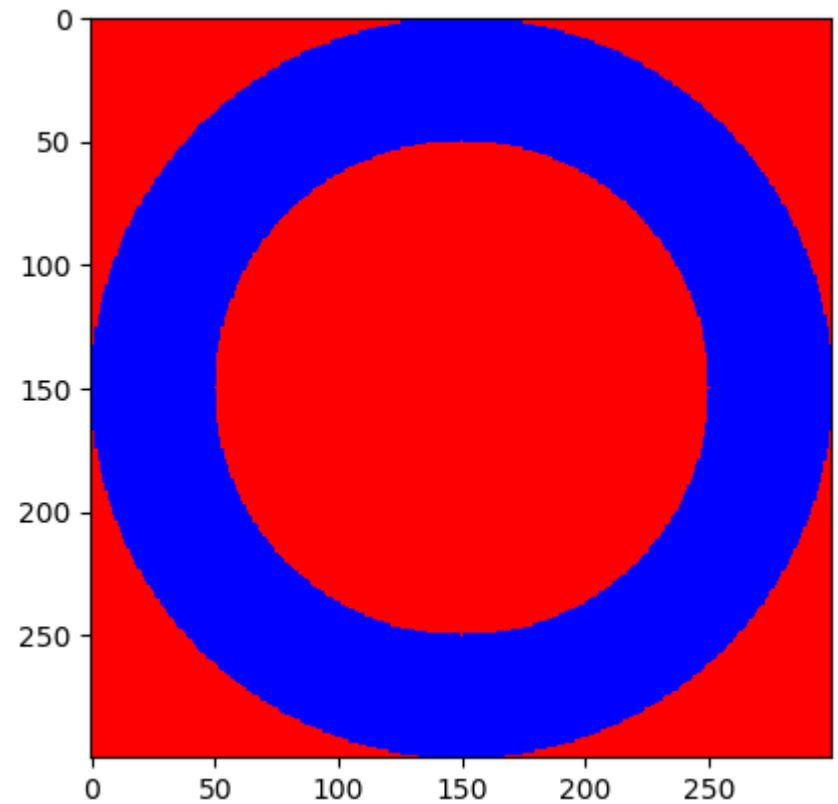
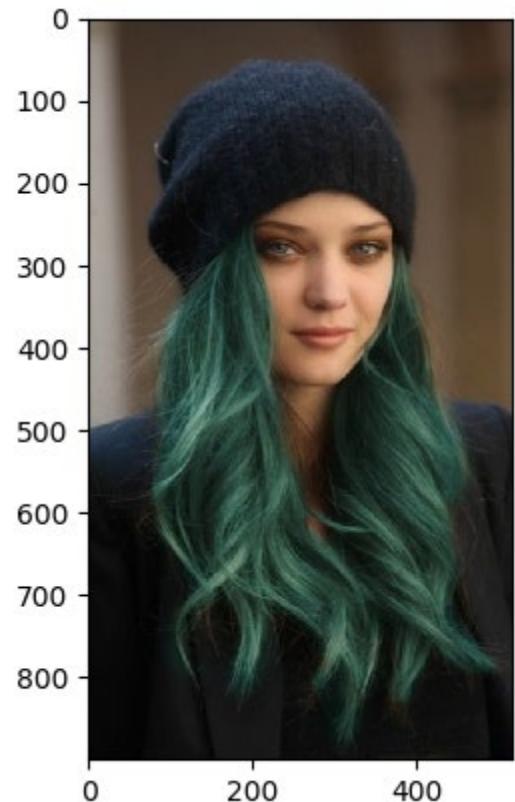


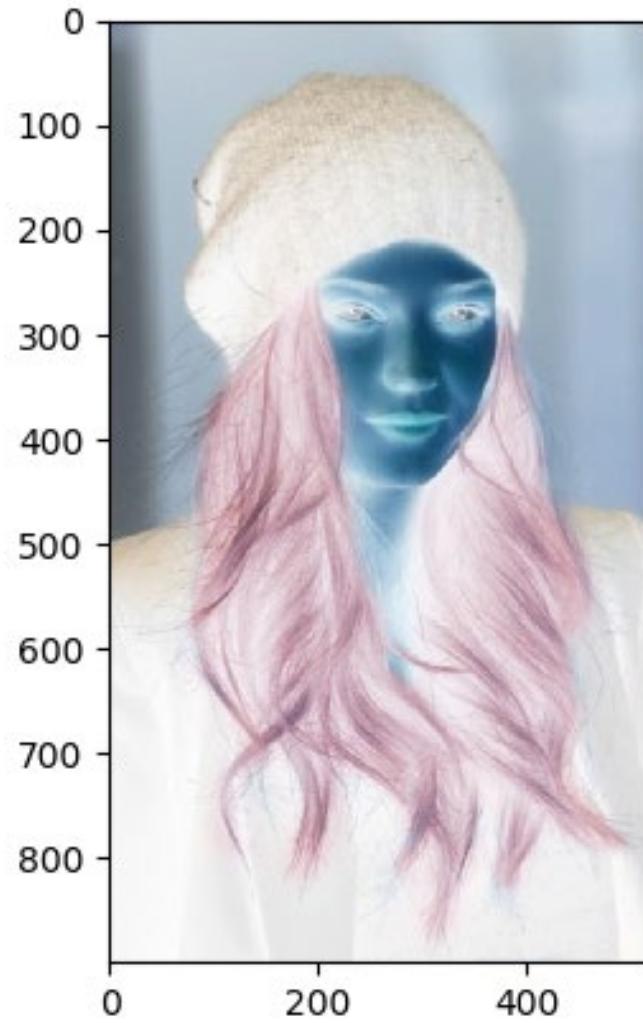
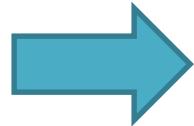
Image Processing

Using the `imageio` package to read in a photo as a 3D

```
>>> import imageio  
>>> import matplotlib.pyplot as plt  
>>> pic = imageio.imread('green hair girl.jpg')  
>>> len(pic)  
900  
>>> len(pic[0])  
517  
>>> len(pic[0][0])  
3  
>>> plt.imshow(pic)  
<matplotlib.image.AxesImage object at  
>>> plt.show()
```



Negative Image

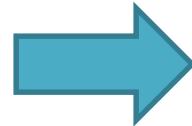


Negative Image

```
import imageio
def negImage(imgFile):
    pic = imageio.imread(imgFile)
    plt.imshow(pic)
    plt.show()          The picture has h x w numbers of pixels
                        (h = #rows, w = #columns)
    h = len(pic)       ← Loop through each pixel with indices (i, j)
    w = len(pic[0])
    for i in range(h):
        for j in range(w):
            for k in range(3):
                pic[i][j][k] = 255 - pic[i][j][k]

    plt.imshow(pic)
    plt.show()
```

Image Processing



Dyeing Hair

```
import imageio

def dye_hair():
    pic = imageio.imread('green hair girl.jpg')
    plt.imshow(pic)
    plt.show()

    h = len(pic)
    w = len(pic[0])
    for i in range(h):
        for j in range(w):
            R = pic[i][j][0]
            G = pic[i][j][1]
            B = pic[i][j][2]
            if G > R and G > B:
                pic[i][j] = [R*2, G*0.2, B*0.8]
    plt.imshow(pic)
    plt.show()
```

Loop through each pixel with indices (i, j)

If `pic[i][j]` has more green, change it to purple

Multi-dimensional Array

$2+1 = 3D$ Array as a Picture

```
pic = [[[0,0,255], [255,0,0], [0,0,255]],  
        [[255,0,0], [255,0,0], [255,0,0]],  
        [[0,0,255], [255,0,0], [0,0,255]]]
```

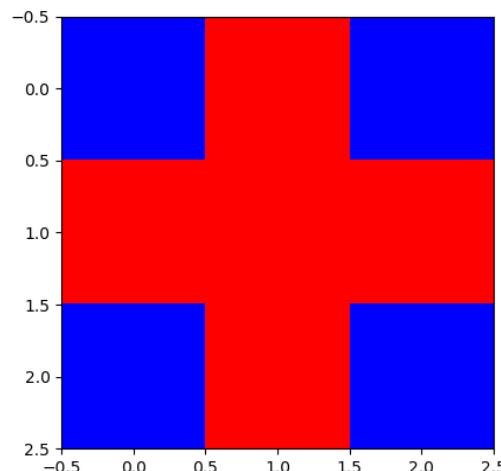
```
import matplotlib.pyplot as plt  
plt.imshow(pic)  
plt.show()
```

One pixel is a list of three values of red, blue and green.

Usually ranging from 0 to 255

`imshow()` is to draw a picture in bitmap

- 3×3 pixels (zoomed)



Dimensions

One Dimensional Array, A

Index	Contents
0	'Apple'
1	'John'
2	'Eve'
3	'Mary'
4	'Ian'
5	'Smith'
6	'Kelvin'

$A[5] = \text{'Smith'}$

Two Dimensional Array, M

	0	1	2	3
0	'Apple'	'Lah'	'Cat'	'Eve'
1	'Hello'	'Pay'	'TV'	'Carl'
2	'What'	'Bank'	'Radio'	'Ada'
3	'Frog'	'Peter'	'Sea'	'Eat'
4	'Job'	'Fry'	'Gym'	'Wow'
5	'Walk'	'Fly'	'Cook'	'Look'

$M[4][1] = \text{'Fry'}$

Multi-Dimensional Array

- A **three** dimensional array

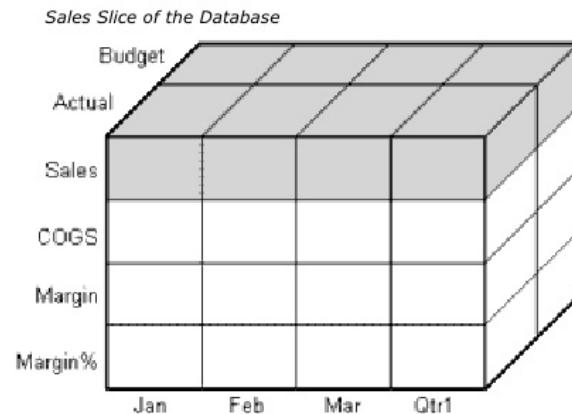
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 128 | 129 | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 140 | 141 | 142 | 143 | 144 | 145 | 146 | 147 | 148 | 149 | 150 | 151 | 152 | 153 | 154 | 155 | 156 | 157 | 158 | 159 | 160 | 161 | 162 | 163 | 164 | 165 | 166 | 167 | 168 | 169 | 170 | 171 | 172 | 173 | 174 | 175 | 176 | 177 | 178 | 179 | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 | 188 | 189 | 190 | 191 | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 | 208 | 209 | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 | 218 | 219 | 220 | 221 | 222 | 223 | 224 | 225 | 226 | 227 | 228 | 229 | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 | 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 248 | 249 | 250 | 251 | 252 | 253 | 254 | 255 | 256 | 257 | 258 | 259 | 260 | 261 | 262 | 263 | 264 | 265 | 266 | 267 | 268 | 269 | 270 | 271 | 272 | 273 | 274 | 275 | 276 | 277 | 278 | 279 | 280 | 281 | 282 | 283 | 284 | 285 | 286 | 287 | 288 | 289 | 290 | 291 | 292 | 293 | 294 | 295 | 296 | 297 | 298 | 299 | 300 | 301 | 302 | 303 | 304 | 305 | 306 | 307 | 308 | 309 | 310 | 311 | 312 | 313 | 314 | 315 | 316 | 317 | 318 | 319 | 320 | 321 | 322 | 323 | 324 | 325 | 326 | 327 | 328 | 329 | 330 | 331 | 332 | 333 | 334 | 335 | 336 | 337 | 338 | 339 | 340 | 341 | 342 | 343 | 344 | 345 | 346 | 347 | 348 | 349 | 350 | 351 | 352 | 353 | 354 | 355 | 356 | 357 | 358 | 359 | 360 | 361 | 362 | 363 | 364 | 365 | 366 | 367 | 368 | 369 | 370 | 371 | 372 | 373 | 374 | 375 | 376 | 377 | 378 | 379 | 380 | 381 | 382 | 383 | 384 | 385 | 386 | 387 | 388 | 389 | 390 | 391 | 392 | 393 | 394 | 395 | 396 | 397 | 398 | 399 | 400 | 401 | 402 | 403 | 404 | 405 | 406 | 407 | 408 | 409 | 410 | 411 | 412 | 413 | 414 | 415 | 416 | 417 | 418 | 419 | 420 | 421 | 422 | 423 | 424 | 425 | 426 | 427 | 428 | 429 | 430 | 431 | 432 | 433 | 434 | 435 | 436 | 437 | 438 | 439 | 440 | 441 | 442 | 443 | 444 | 445 | 446 | 447 | 448 | 449 | 450 | 451 | 452 | 453 | 454 | 455 | 456 | 457 | 458 | 459 | 460 | 461 | 462 | 463 | 464 | 465 | 466 | 467 | 468 | 469 | 470 | 471 | 472 | 473 | 474 | 475 | 476 | 477 | 478 | 479 | 480 | 481 | 482 | 483 | 484 | 485 | 486 | 487 | 488 | 489 | 490 | 491 | 492 | 493 | 494 | 495 | 496 | 497 | 498 | 499 | 500 | 501 | 502 | 503 | 504 | 505 | 506 | 507 | 508 | 509 | 510 | 511 | 512 | 513 | 514 | 515 | 516 | 517 | 518 | 519 | 520 | 521 | 522 | 523 | 524 | 525 | 526 | 527 | 528 | 529 | 530 | 531 | 532 | 533 | 534 | 535 | 536 | 537 | 538 | 539 | 540 | 541 | 542 | 543 | 544 | 545 | 546 | 547 | 548 | 549 | 550 | 551 | 552 | 553 | 554 | 555 | 556 | 557 | 558 | 559 | 560 | 561 | 562 | 563 | 564 | 565 | 566 | 567 | 568 | 569 | 570 | 571 | 572 | 573 | 574 | 575 | 576 | 577 | 578 | 579 | 580 | 581 | 582 | 583 | 584 | 585 | 586 | 587 | 588 | 589 | 590 | 591 | 592 | 593 | 594 | 595 | 596 | 597 | 598 | 599 | 600 | 601 | 602 | 603 | 604 | 605 | 606 | 607 | 608 | 609 | 610 | 611 | 612 | 613 | 614 | 615 | 616 | 617 | 618 | 619 | 620 | 621 | 622 | 623 | 624 | 625 | 626 | 627 | 628 | 629 | 630 | 631 | 632 | 633 | 634 | 635 | 636 | 637 | 638 | 639 | 640 | 641 | 642 | 643 | 644 | 645 | 646 | 647 | 648 | 649 | 650 | 651 | 652 | 653 | 654 | 655 | 656 | 657 | 658 | 659 | 660 | 661 | 662 | 663 | 664 | 665 | 666 | 667 | 668 | 669 | 670 | 671 | 672 | 673 | 674 | 675 | 676 | 677 | 678 | 679 | 680 | 681 | 682 | 683 | 684 | 685 | 686 | 687 | 688 | 689 | 690 | 691 | 692 | 693 | 694 | 695 | 696 | 697 | 698 | 699 | 700 | 701 | 702 | 703 | 704 | 705 | 706 | 707 | 708 | 709 | 710 | 711 | 712 | 713 | 714 | 715 | 716 | 717 | 718 | 719 | 720 | 721 | 722 | 723 | 724 | 725 | 726 | 727 | 728 | 729 | 730 | 731 | 732 | 733 | 734 | 735 | 736 | 737 | 738 | 739 | 740 | 741 | 742 | 743 | 744 | 745 | 746 | 747 | 748 | 749 | 750 | 751 | 752 | 753 | 754 | 755 | 756 | 757 | 758 | 759 | 760 | 761 | 762 | 763 | 764 | 765 | 766 | 767 | 768 | 769 | 770 | 771 | 772 | 773 | 774 | 775 | 776 | 777 | 778 | 779 | 780 | 781 | 782 | 783 | 784 | 785 | 786 | 787 | 788 | 789 | 790 | 791 | 792 | 793 | 794 | 795 | 796 | 797 | 798 | 799 | 800 | 801 | 802 | 803 | 804 | 805 | 806 | 807 | 808 | 809 | 810 | 811 | 812 | 813 | 814 | 815 | 816 | 817 | 818 | 819 | 820 | 821 | 822 | 823 | 824 | 825 | 826 | 827 | 828 | 829 | 830 | 831 | 832 | 833 | 834 | 835 | 836 | 837 | 838 | 839 | 840 | 841 | 842 | 843 | 844 | 845 | 846 | 847 | 848 | 849 | 850 | 851 | 852 | 853 | 854 | 855 | 856 | 857 | 858 | 859 | 860 | 861 | 862 | 863 | 864 | 865 | 866 | 867 | 868 | 869 | 870 | 871 | 872 | 873 | 874 | 875 | 876 | 877 | 878 | 879 | 880 | 881 | 882 | 883 | 884 | 885 | 886 | 887 | 888 | 889 | 890 | 891 | 892 | 893 | 894 | 895 | 896 | 897 | 898 | 899 | 900 | 901 | 902 | 903 | 904 | 905 | 906 | 907 | 908 | 909 | 910 | 911 | 912 | 913 | 914 | 915 | 916 | 917 | 918 | 919 | 920 | 921 | 922 | 923 | 924 | 925 | 926 | 927 | 928 | 929 | 930 | 931 | 932 | 933 | 934 | 935 | 936 | 937 | 938 | 939 | 940 | 941 | 942 | 943 | 944 | 945 | 946 | 947 | 948 | 949 | 950 | 951 | 952 | 953 | 954 | 955 | 956 | 957 | 958 | 959 | 960 | 961 | 962 | 963 | 964 | 965 | 966 | 967 | 968 | 969 | 970 | 971 | 972 | 973 | 974 | 975 | 976 | 977 | 978 | 979 | 980 | 981 | 982 | 983 | 984 | 985 | 986 | 987 | 988 | 989 | 990 | 991 | 992 | 993 | 994 | 995 | 996 | 997 | 998 | 999 | 1000 | 1001 | 1002 | 1003 | 1004 | 1005 | 1006 | 1007 | 1008 | 1009 | 1010 | 1011 | 1012 | 1013 | 1014 | 1015 | 1016 | 1017 | 1018 | 1019 | 1020 | 1021 | 1022 | 1023 | 1024 | 1025 | 1026 | 1027 | 1028 | 1029 | 1030 | 1031 | 1032 | 1033 | 1034 | 1035 | 1036 | 1037 | 1038 | 1039 | 1040 | 1041 | 1042 | 1043 | 1044 | 1045 | 1046 | 1047 | 1048 | 1049 | 1050 | 1051 | 1052 | 1053 | 1054 | 1055 | 1056 | 1057 | 1058 | 1059 | 1060 | 1061 | 1062 | 1063 | 1064 | 1065 | 1066 | 1067 | 1068 | 1069 | 1070 | 1071 | 1072 | 1073 | 1074 | 1075 | 1076 | 1077 | 1078 | 1079 | 1080 | 1081 | 1082 | 1083 | 1084 | 1085 | 1086 | 1087 | 1088 | 1089 | 1090 | 1091 | 1092 | 1093 | 1094 | 1095 | 1096 | 1097 | 1098 | 1099 | 1100 | 1101 | 1102 | 1103 | 1104 | 1105 | 1106 | 1107 | 1108 | 1109 | 1110 | 1111 | 1112 | 1113 | 1114 | 1115 | 1116 | 1117 | 1118 | 1119 | 1120 | 1121 | 1122 | 1123 | 1124 | 1125 | 1126 | 1127 | 1128 | 1129 | 1130 | 1131 | 1132 | 1133 | 1134 | 1135 | 1136 | 1137 | 1138 | 1139 | 1140 | 1141 | 1142 | 1143 | 1144 | 1145 | 1146 | 1147 | 1148 | 1149 | 1150 | 1151 | 1152 | 1153 | 1154 | 1155 | 1156 | 1157 | 1158 | 1159 | 1160 | 1161 | 1162 | 1163 | 1164 | 1165 | 1166 | 1167 | 1168 | 1169 | 1170 | 1171 | 1172 | 1173 | 1174 | 1175 | 1176 | 1177 | 1178 | 1179 | 1180 | 1181 | 1182 | 1183 | 1184 | 1185 | 1186 | 1187 | 1188 | 1189 | 1190 | 1191 | 1192 | 1193 | 1194 | 1195 | 1196 | 1197 | 1198 | 1199 | 1200 | 1201 | 1202 | 1203 | 1204 | 1205 | 1206 | 1207 | 1208 | 1209 | 1210 | 1211 | 1212 | 1213 | 1214 | 1215 | 1216 | 1217 | 1218 | 1219 | 1220 | 1221 | 1222 | 1223 | 1224 | 1225 | 1226 | 1227 | 1228 | 1229 | 1230 | 1231 | 1232 | 1233 | 1234 | 1235 | 1236 | 1237 | 1238 | 1239 | 1240 | 1241 | 1242 | 1243 | 1244 | 1245 | 1246 | 1247 | 1248 | 1249 | 1250 | 1251 | 1252 | 1253 | 1254 | 1255 | 1256 | 1257 | 1258 | 1259 | 1260 | 1261 | 1262 | 1263 | 1264 | 1265 | 1266 | 1267 | 1268 | 1269 | 1270 | 1271 | 1272 | 1273 | 1274 | 1275 | 1276 | 1277 | 1278 | 1279 | 1280 | 1281 | 1282 | 1283 | 1284 | 1285 | 1286 | 1287 | 1288 | 1289 | 1290 | 1291 | 1292 | 1293 | 1294 | 1295 | 1296 | 1297 | 1298 | 1299 | 1300 | 1301 | 1302 | 1303 | 1304 | 1305 | 1306 | 1307 | 1308 | 1309 | 1310 | 1311 | 1312 | 1313 | 1314 | 1315 | 1316 | 1317 | 1318 | 1319 | 1320 | 1321 | 1322 | 1323 | 1324 | 1325 | 1326 | 1327 | 1328 | 1329 | 1330 | 1331 | 1332 | 1333 | 1334 | 1335 | 1336 | 1337 | 1338 | 1339 | 1340 | 1341 | 1342 | 1343 | 1344 | 1345 | 1346 | 1347 | 1348 | 1349 | 1350 | 1351 | 1352 | 1353 | 1354 | 1355 | 1356 | 1357 | 1358 | 1359 | 1360 | 1361 | 1362 | 1363 | 1364 | 1365 | 1366 | 1367 | 1368 | 1369 | 1370 | 1371 | 1372 | 1373 | 1374 | 1375 | 1376 | 1377 | 1378 | 1379 | 1380 | 1381 | 1382 | 1383 | 1384 | 1385 | 1386 | 1387 | 1388 | 1389 | 1390 | 1391 | 1392 | 1393 | 1394 | 1395 | 1396 | 1397 | 1398 | 1399 | 1400 | 1401 | 1402 | 1403 | 1404 | 1405 | 1406 | 1407 | 1408 | 1409 | 1410 | 1411 | 1412 | 1413 | 1414 | 1415 | 1416 | 1417 | 1418 | 1419 | 1420 | 1421 | 1422 | 1423 | 1424 | 1425 | 1426 | 1427 | 1428 | 1429 | 1430 | 1431 | 1432 | 1433 | 1434 | 1435 | 1436 | 1437 | 1438 | 1439 | 1440 | 1441 | 1442 | 1443 | 1444 | 1445 | 1446 | 1447 | 1448 | 1449 | 1450 | 1451 | 1452 | 1453 | 1454 | 1455 | 1456 | 1457 | 1458 | 1459 | 1460 | 1461 | 1462 |<
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Fancy Terminology in Business

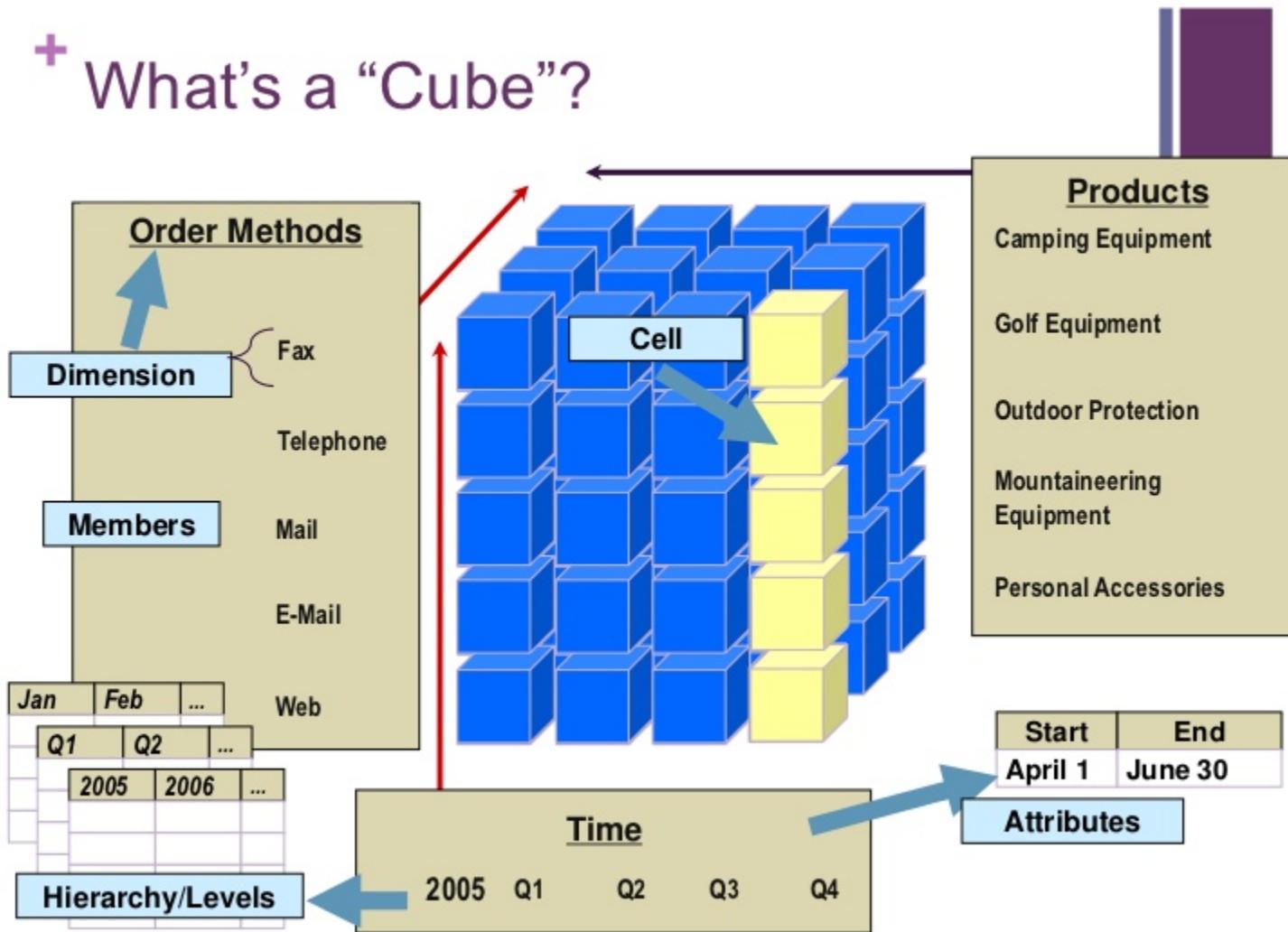
- Multidimensional Data Model
- Multidimensional Analysis



The shaded cells is called a slice illustrate that, when you refer to Sales, you are referring to the portion of the database containing eight Sales values.

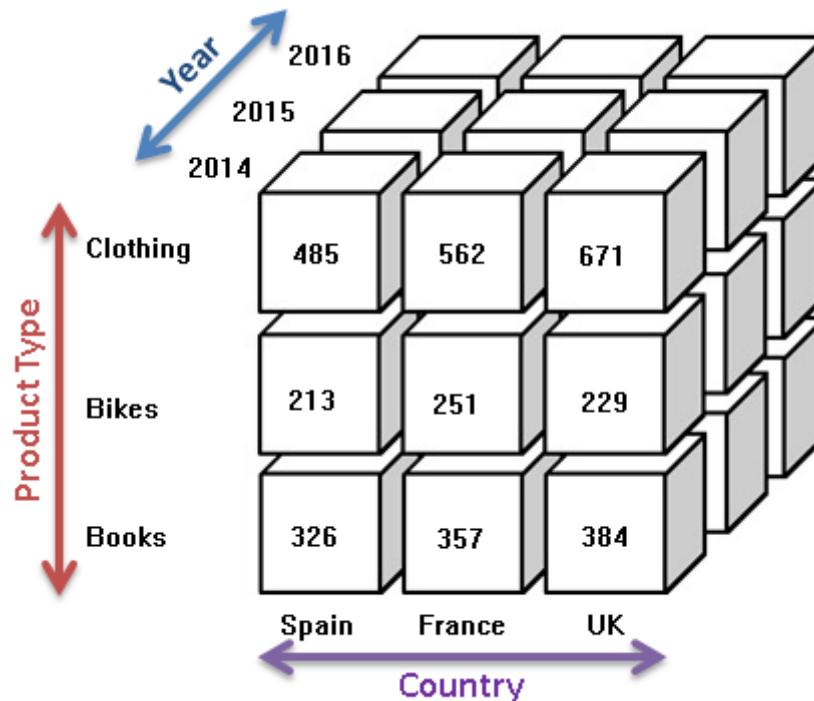


+ What's a “Cube”?



Sales Cube

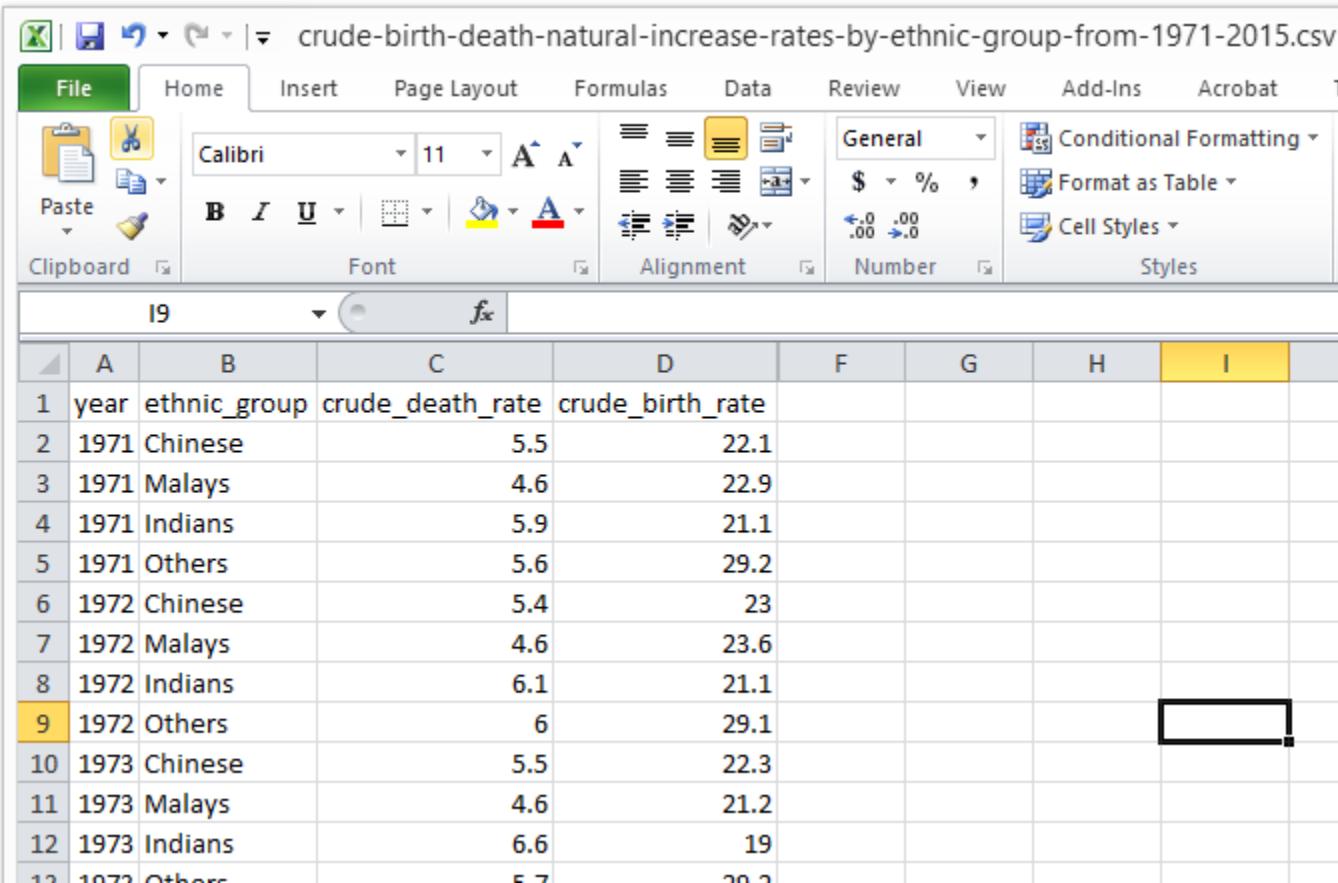
- Product Type x Year x Country



- 4th Dimensional Sales Cube:
 - Product Type x Year x Country x {Predicted vs actual}

Multi-Dimensional Array

- Year x Ethnic Group x {death/birth}



The screenshot shows a Microsoft Excel spreadsheet titled "crude-birth-death-natural-increase-rates-by-ethnic-group-from-1971-2015.csv". The data is presented in a table with columns labeled A through I. The first few rows of data are as follows:

	A	B	C	D	F	G	H	I
1	year	ethnic_group	crude_death_rate	crude_birth_rate				
2	1971	Chinese	5.5	22.1				
3	1971	Malays	4.6	22.9				
4	1971	Indians	5.9	21.1				
5	1971	Others	5.6	29.2				
6	1972	Chinese	5.4	23				
7	1972	Malays	4.6	23.6				
8	1972	Indians	6.1	21.1				
9	1972	Others	6	29.1				
10	1973	Chinese	5.5	22.3				
11	1973	Malays	4.6	21.2				
12	1973	Indians	6.6	19				
13	1973	Others	5.7	29.2				

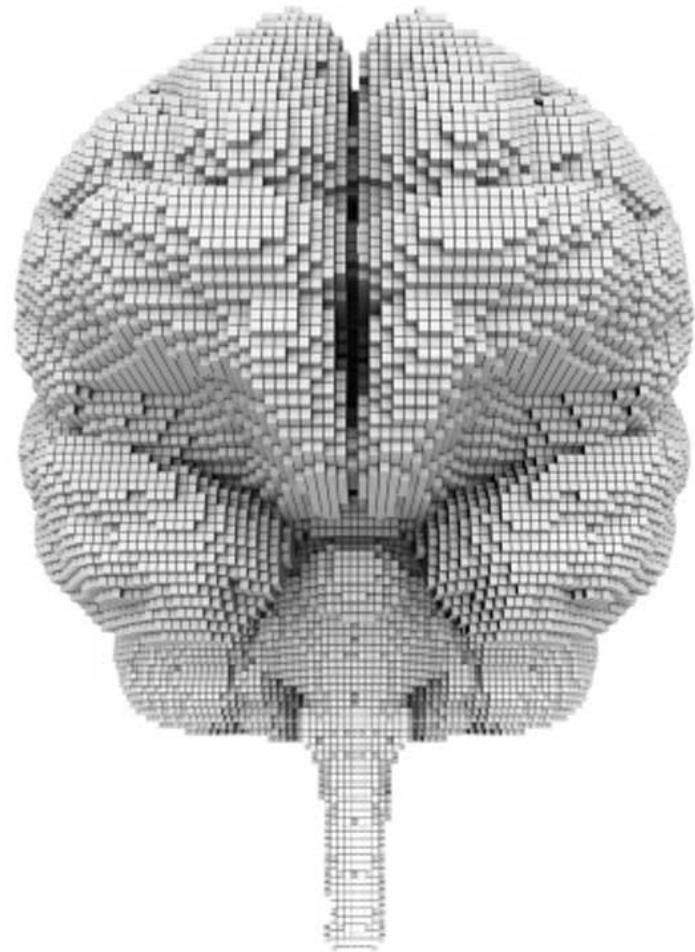
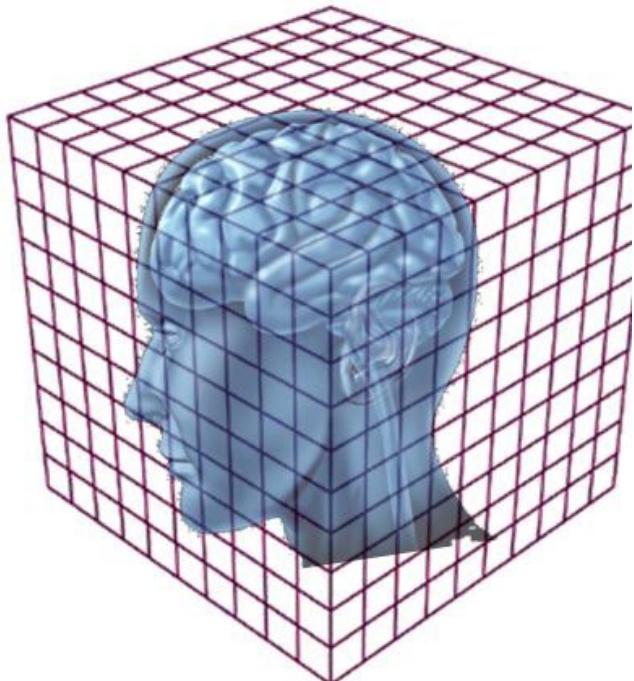
And.... how can we forget?!





Voxels

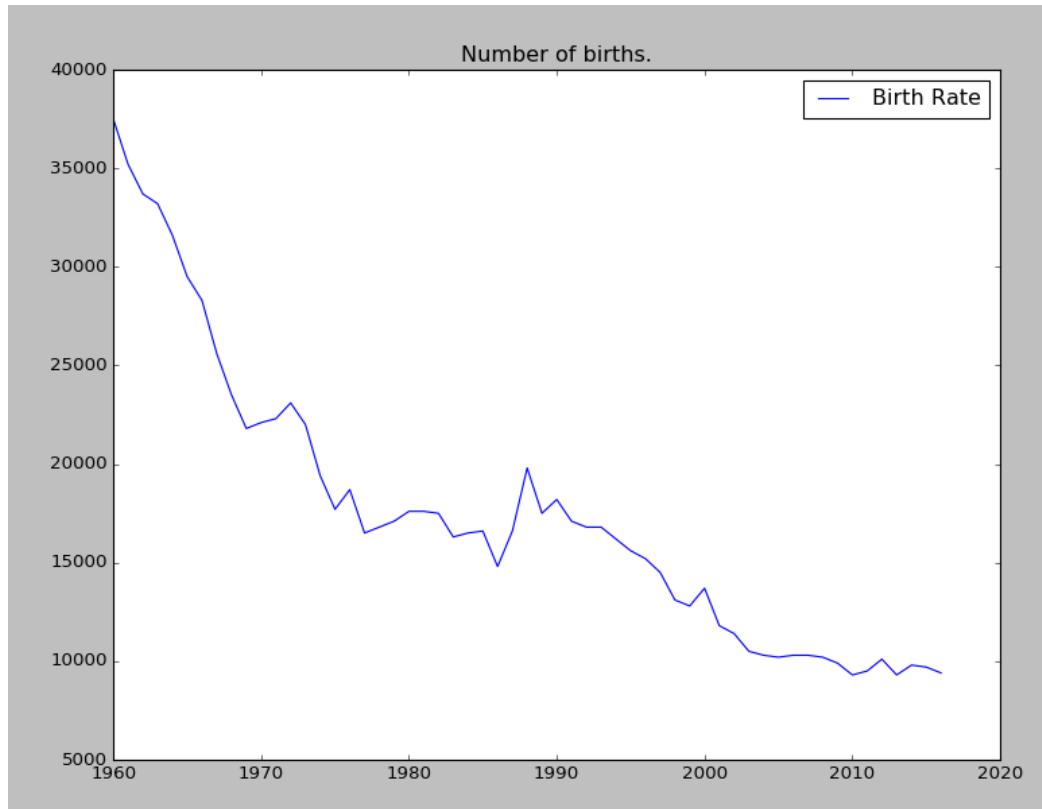
- 2D element: Pixels
- 3D element: Voxels



Reading a CSV File

2D Array x CSV

- Remember last time, we plotted the Singapore birth rate with the data in a CSV file



```
import matplotlib.pyplot as plt

def plot_birth_rate():
    with open('crude-birth-rate.csv') as f:
        f.readline()
        year = []
        num_birth = []
        for line in f:
            list_form = line.rstrip('\n').split(',')
            year.append(int(list_form[0]))
            num_birth.append(float(list_form[2])*1000)

    plt.plot(year, num_birth, label="Birth Rate")
    plt.legend(loc="upper right")
    plt.title('Number of births.')
    plt.show()

plot_birth_rate()
```

Reading CSV Files

Read in a
CSV file
into a list

Create a CSV File
Reader

```
>>> from pprint import pprint
>>> birth_file = open('crude-birth-rate.csv')
>>> birth_file_reader = csv.reader(birth_file)
>>> birth_data = list(birth_file_reader)
```

```
>>> pprint(birth_data)
[['year', 'level_1', 'value'],
 ['1960', 'Crude Birth Rate', '37.5'],
 ['1961', 'Crude Birth Rate', '35.2'],
 ['1962', 'Crude Birth Rate', '33.7'],
 ['1963', 'Crude Birth Rate', '33.2'],
 ['1964', 'Crude Birth Rate', '31.6'],
 ['1965', 'Crude Birth Rate', '29.5'],
 ['1966', 'Crude Birth Rate', '28.3'],
 ['1967', 'Crude Birth Rate', '25.6'],
 ['1968', 'Crude Birth Rate', '23.5'],
 ['1969', 'Crude Birth Rate', '21.8']]
```

Remember these
four lines of code

No need for all
those string
strip(), split() etc.

Caution

- Again, using lists or tuples?
- Again, the old list referencing problem

```
>>> row = [1, 2, 3]
>>> m = []
>>> m.append(row)
>>> m.append(row)
>>> m.append(row)
>>> m
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
>>> m[0][0] = 999
>>> m
[[999, 2, 3], [999, 2, 3], [999, 2, 3]]
>>> m2 = m
>>> m2
[[999, 2, 3], [999, 2, 3], [999, 2, 3]]
>>> m[2][2] = 'X'
>>> m2
[[999, 2, 'X'], [999, 2, 'X'], [999, 2, 'X']]
```



Conclusion

- Dimensionality is not that difficult
 - If you can move from 1D to 2D, moving to 3D or ND is not that difficult
 - *In bosonic string theory, spacetime is 26-dimensional, while in superstring theory it is 10-dimensional, and in M-theory it is 11-dimensional.*
- MD-arrays solve a lot of problems
 - Math, images, engineering, robotics, AI,