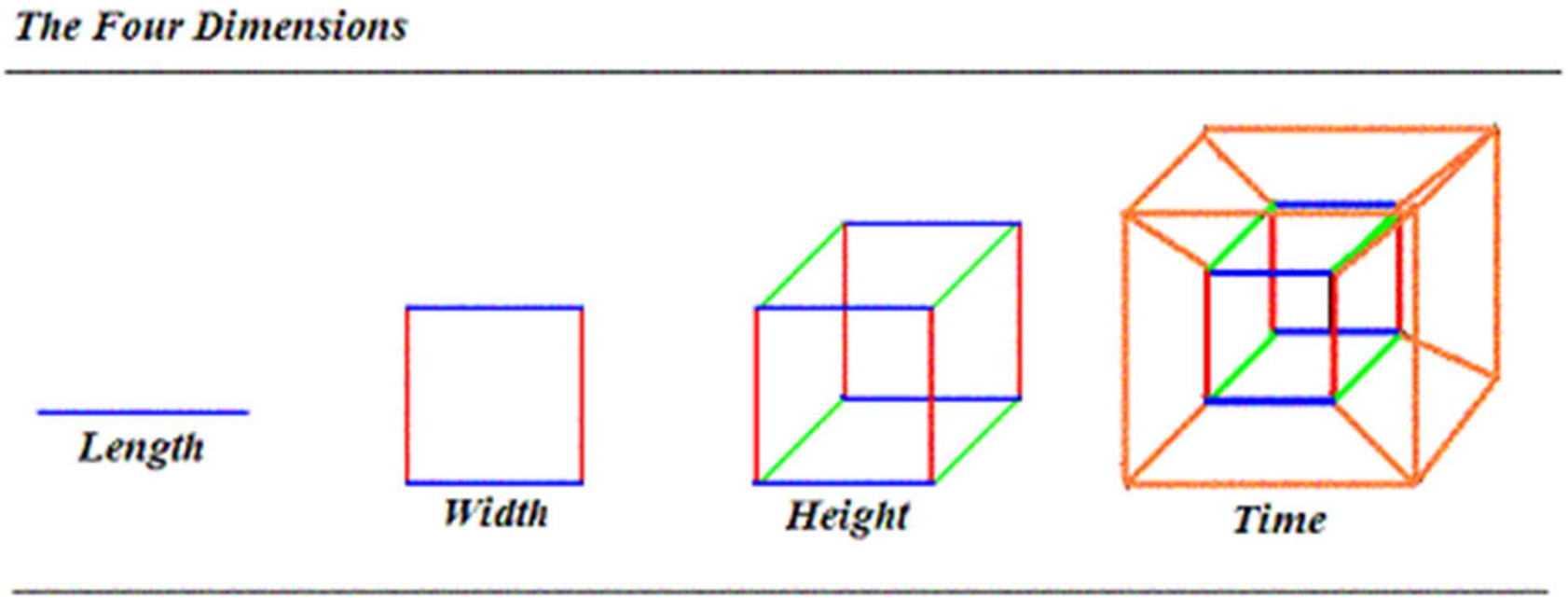
# Dimensions

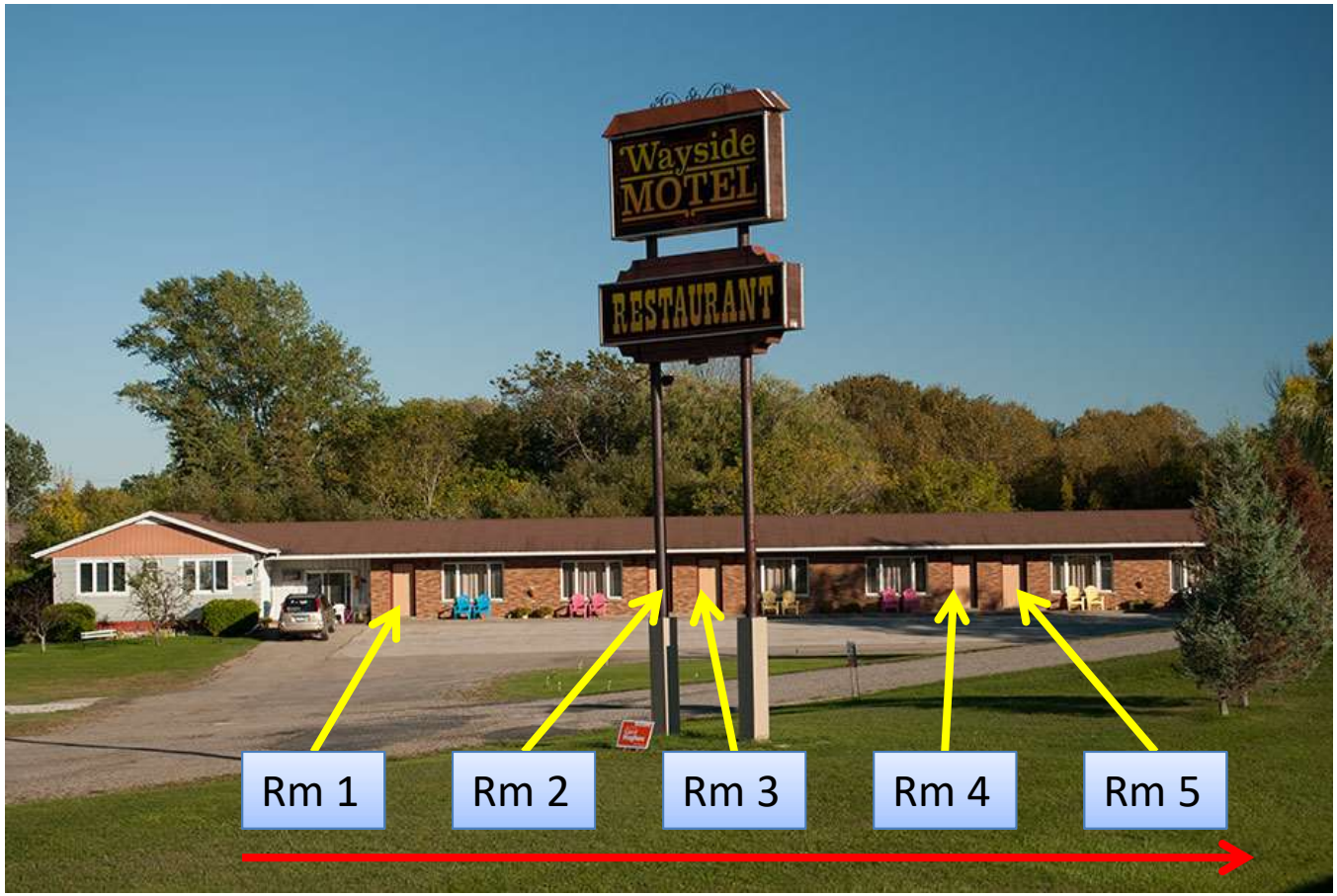
- Are we living in a three dimensional space?




[The xkcd guide to the fourth dimension]

# Two-dimensional Array

# Motel in US (1D)

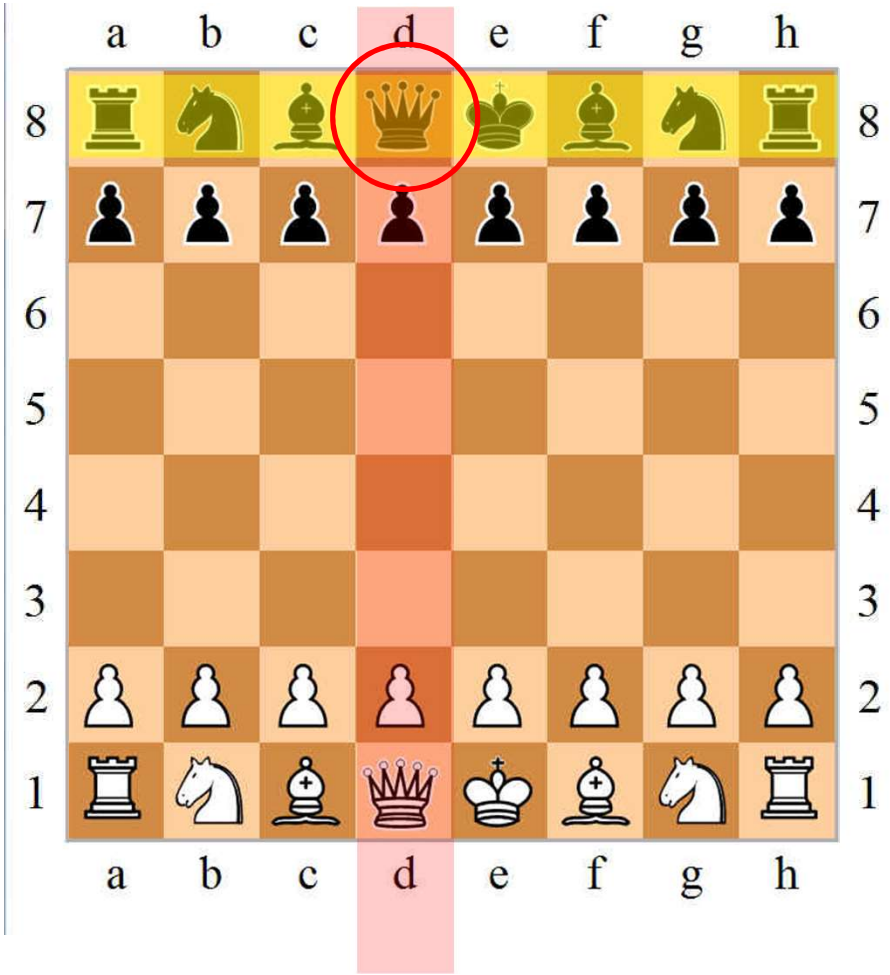


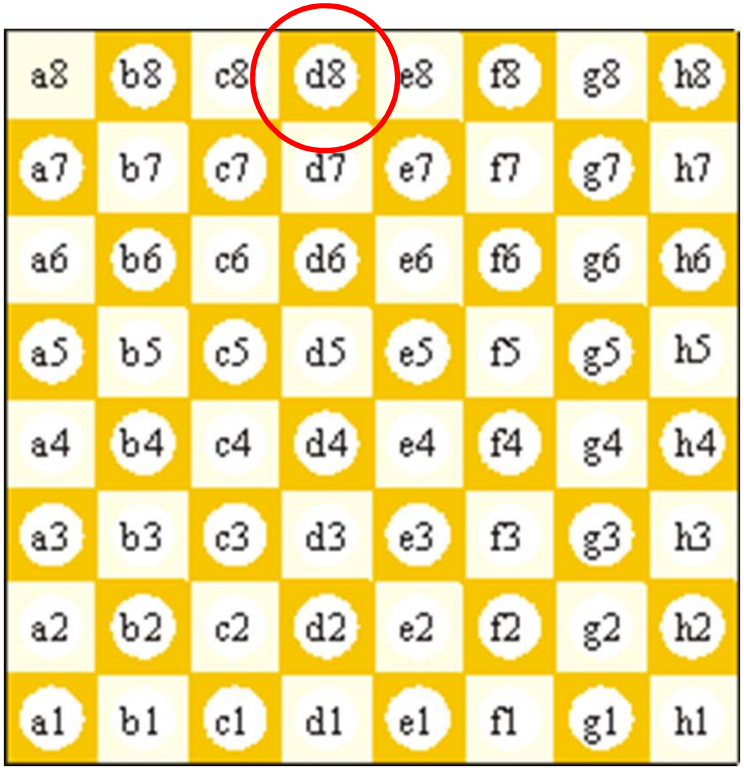

One Dimension

# Hotel (2D)

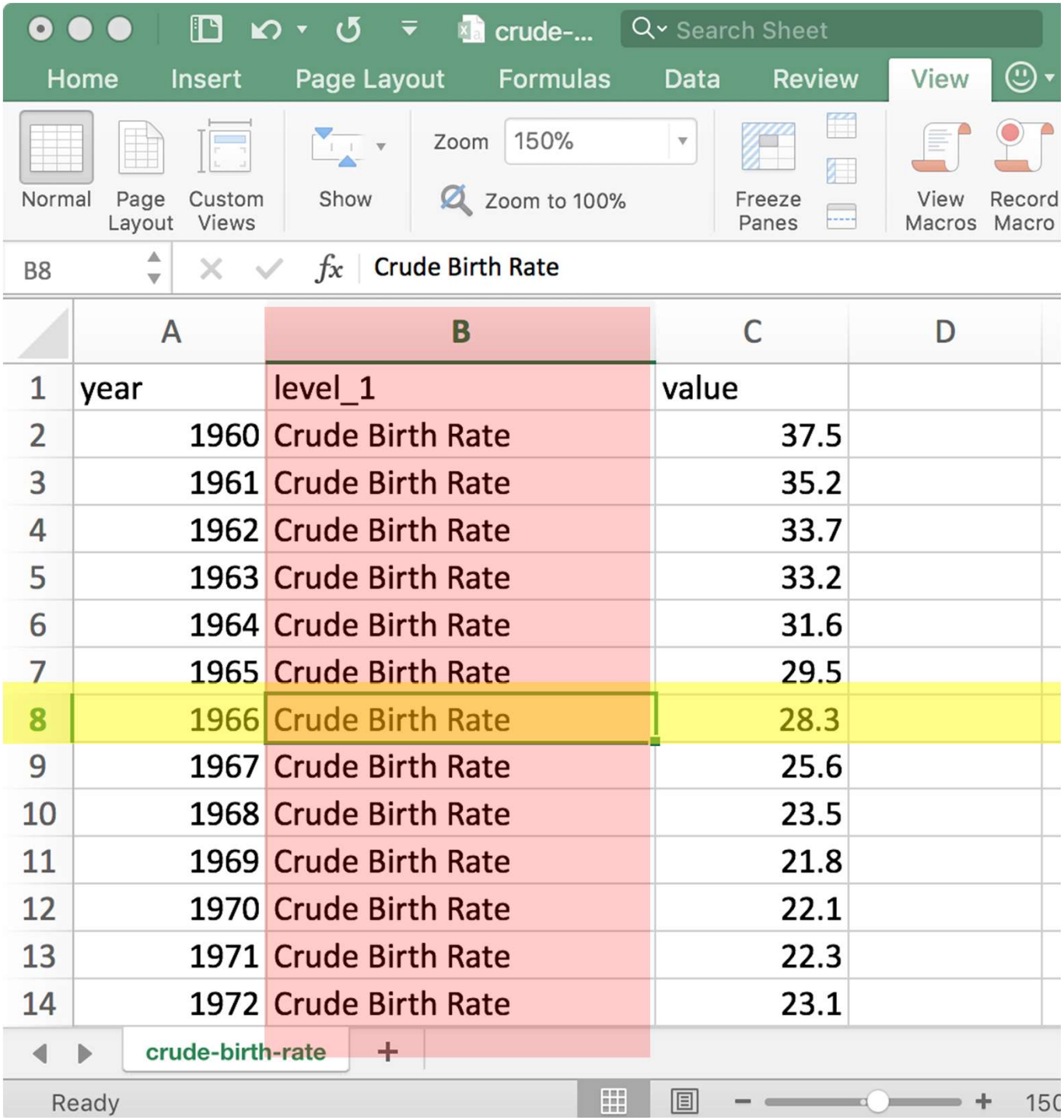
My room is 02-05

# Chess Position



- 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’

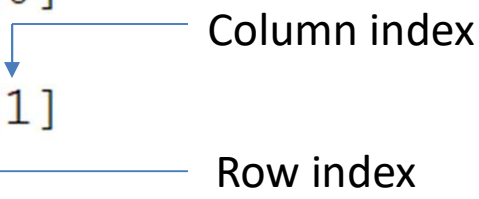
# 2-Dimensional Array in Python

- 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
```

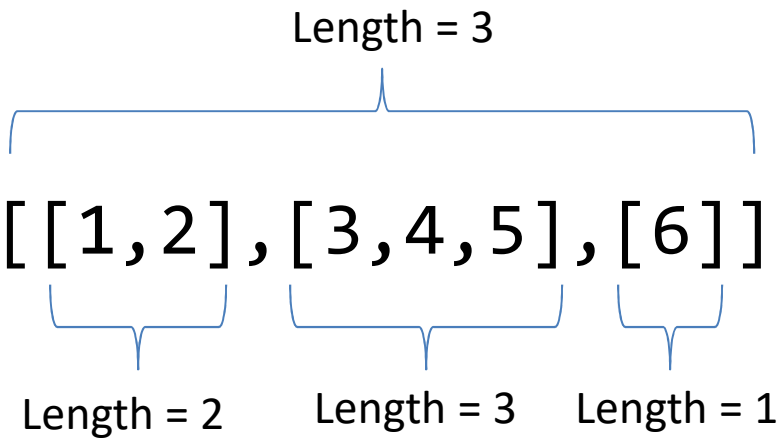



- Equivalent to

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

# What is the Difference Between …

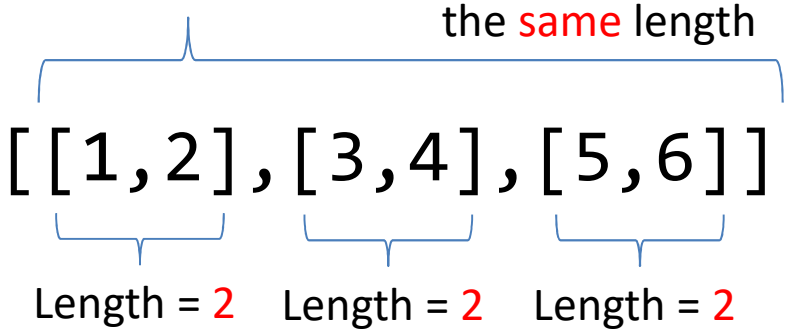
- A list of lists
  - More specifically, all the lists do not have to be the same length




- 2D Array
  - Is a *subset* of a list of lists

#rows,
Can be any length

Can be any length, but **all** the lists have to be the same length




↕

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

# A Larger 2-d Array

- Creating a 4 x 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)
[[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)
4
>>> len(m2[0])
10
```

`pprint()` from the package `pprint` makes the format nicer

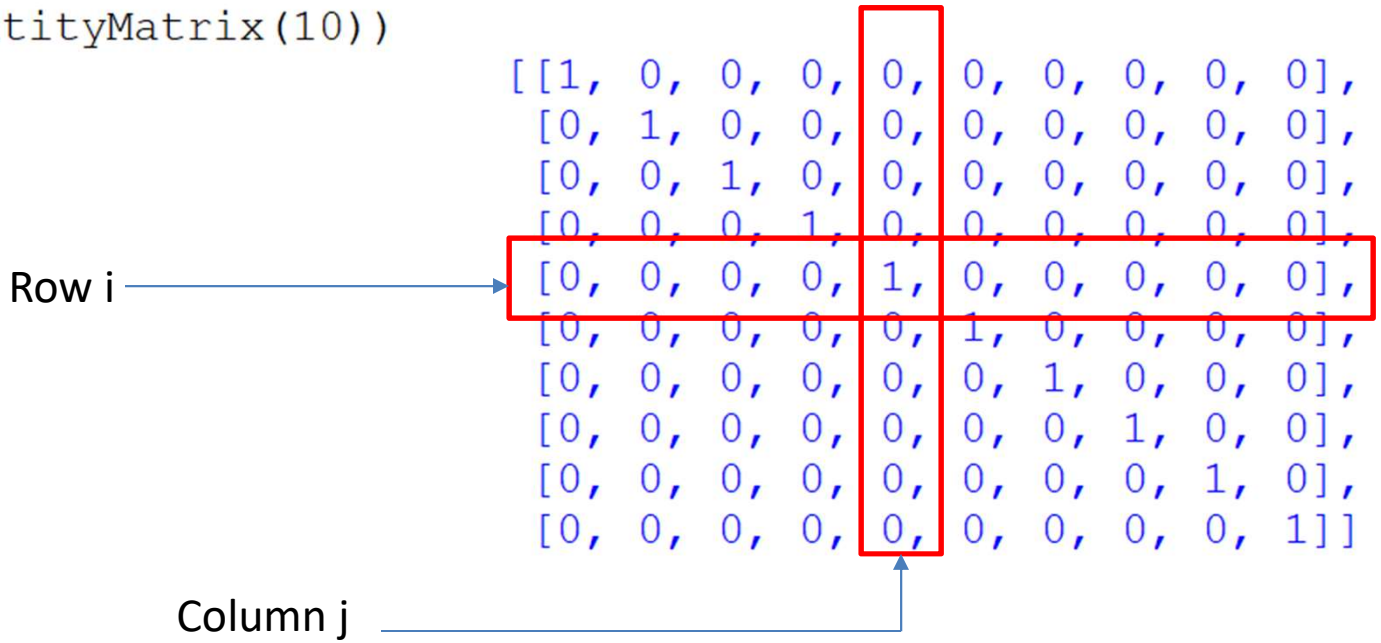
Number of rows

Number of columns

# To Create an N x 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, 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]]
```



# 2D Looping

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

```
[[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]]
```

```
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, 0, 0, 0, 0]]
```

# Accessing Entries

- Remember
  - Frist row index
  - Then column index

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

i = 1 →

↑
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

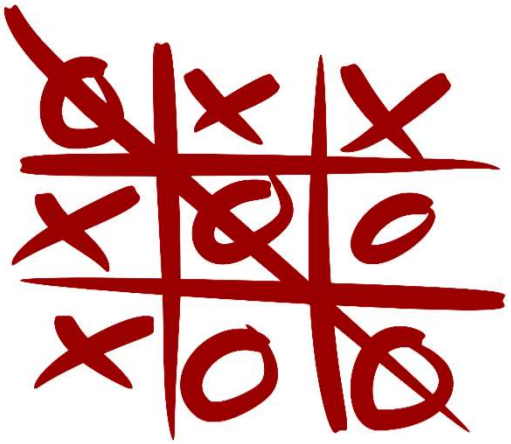
- Or do any computations on the table
  - E.g. The average of the English scores

```
>>> 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 x 3 grid
  - How to represent the game?
    - A 3 x 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

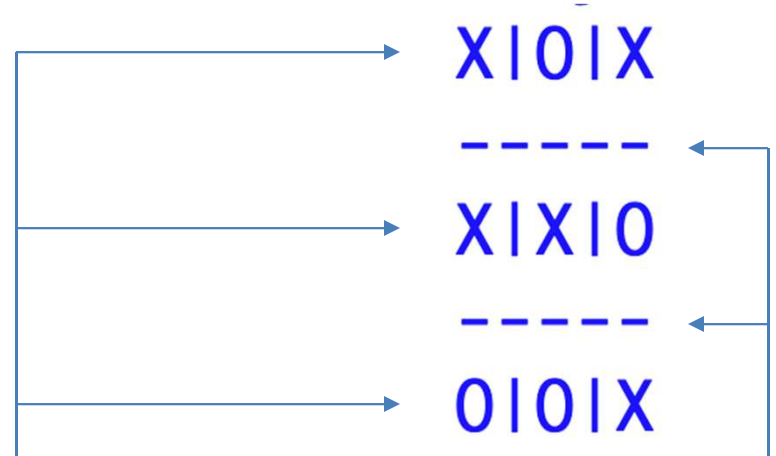
```
i = (pos-1)//3
j = (pos-1)%3
```

```
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

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

```
1|2|3
-----
4|5|6
-----
7|8|9
```

```
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)
```

# 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

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

```
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)
```

# Gameplay

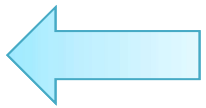
```
1|2|3
-----
4|5|6
-----
7|8|9

Player X move:5
1|2|3
-----
4|X|6
-----
7|8|9
```

```
Player O move:3
1|2|O
-----
4|X|6
-----
7|8|9

Player X move:6
1|2|O
-----
4|X|X
-----
7|8|9
```

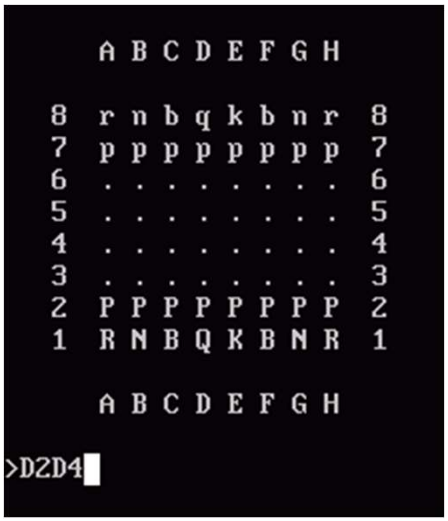
# Chess

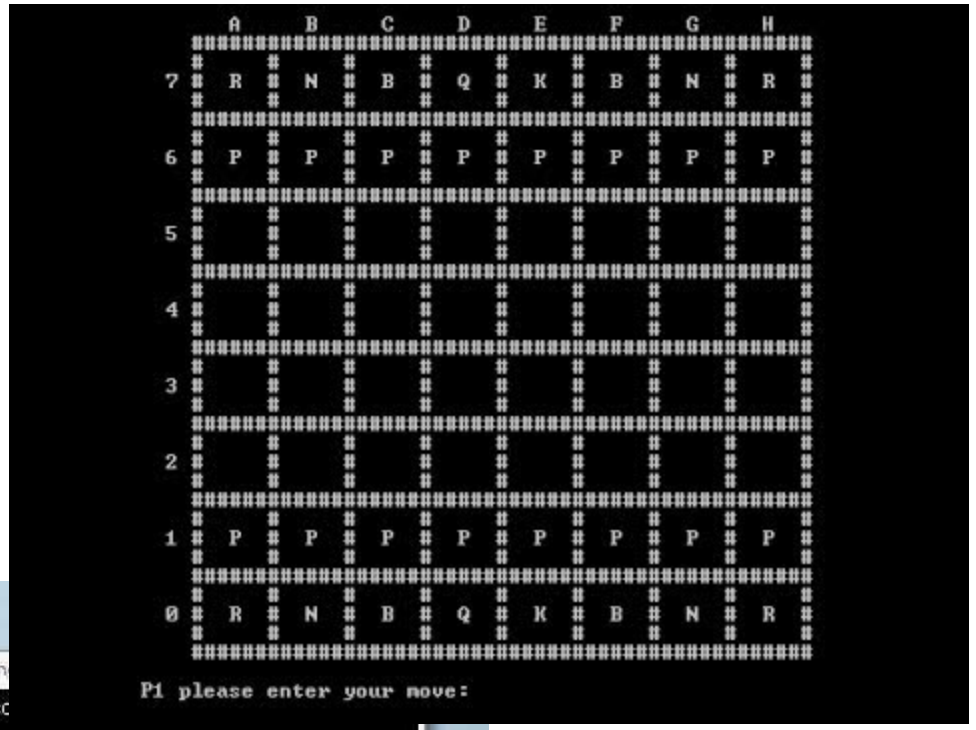
```
def createChessGame():
    game = createZeroMatrix(8,8)
    for j in range(8):
        game[1][j] = 'p'
        game[6][j] = 'P'
    game[7] = ['R','K','B','Q','E','B','K','R']
    game[0] = list(map(lambda x:x.lower(),game[7]))
    return game

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

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



# Other Text Based Chess

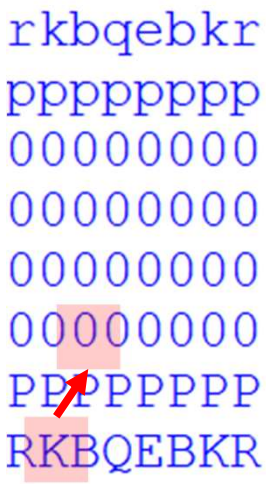


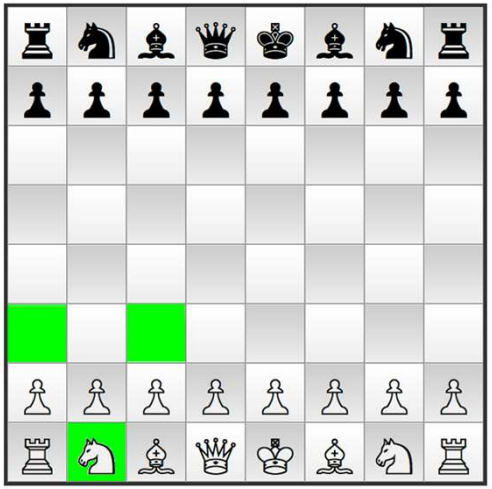




# Chess

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

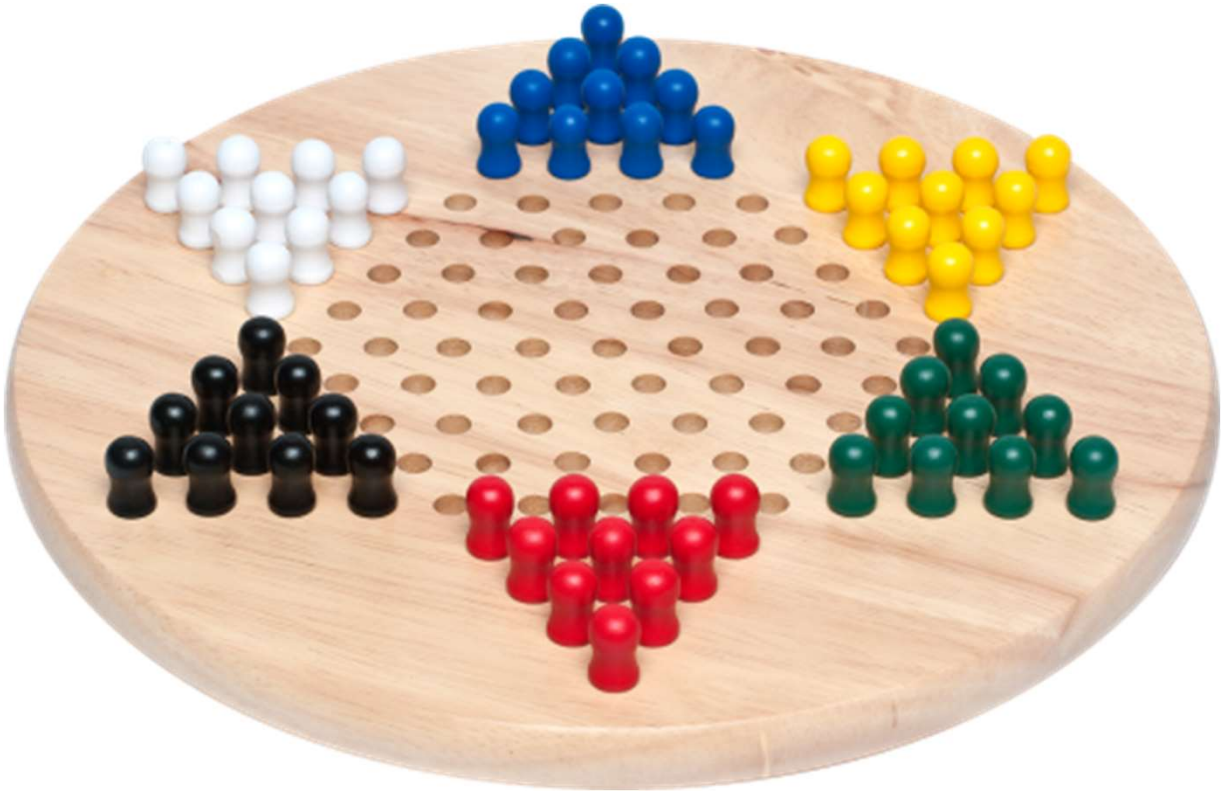
```
game[7][1] = 0
game[5][2] = ‘K’
```



# Chinese Checkers?

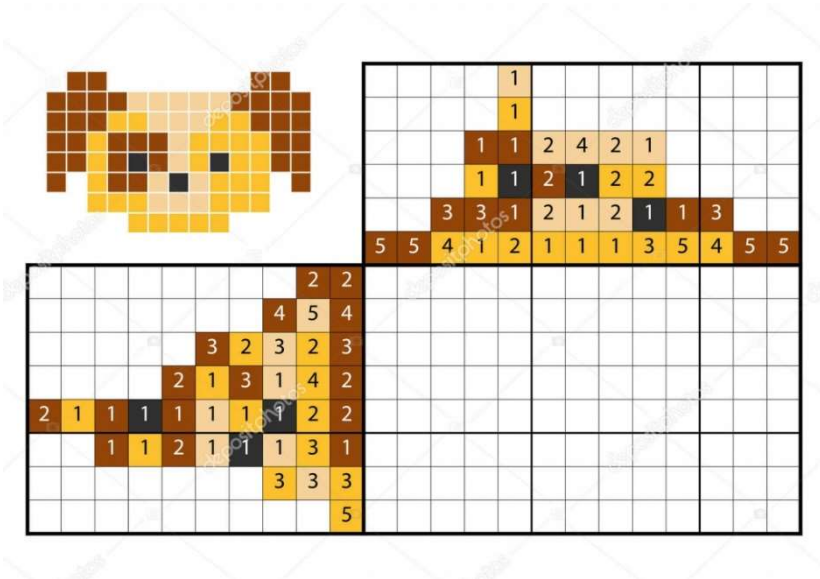
- How to represent the board?

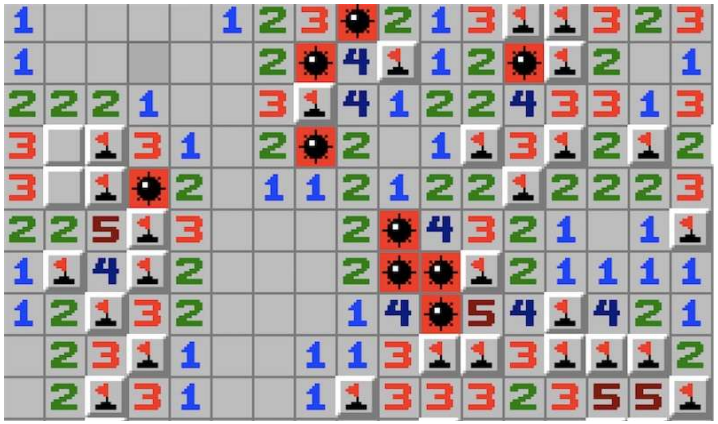


# 3D Chess

# 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 |



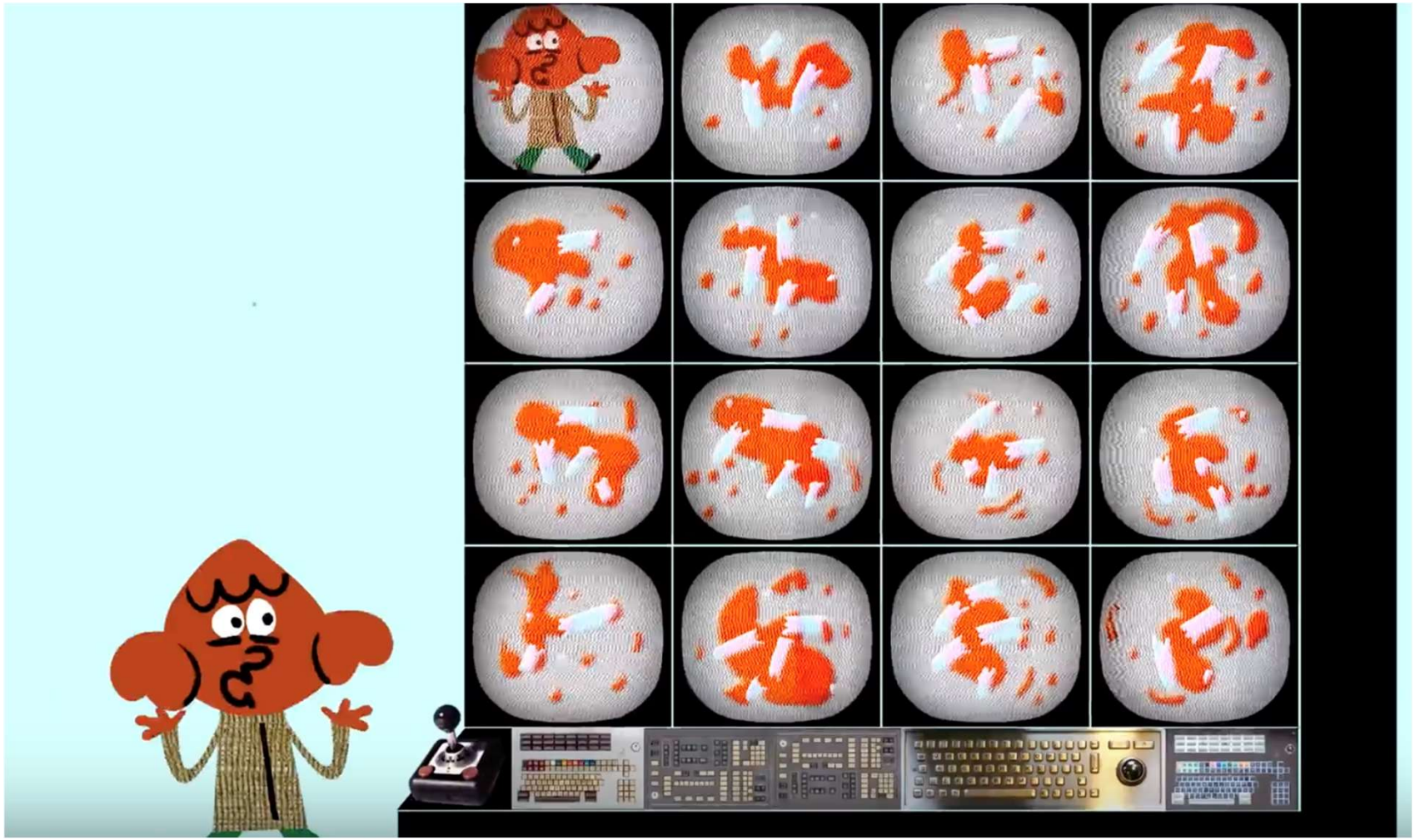


# RPG, SRPG


ファイル(F)　デバッグ(O)　マップ(M)
平地
回避　0
デューン
ハンター
HP　21 / 21
MENU
Desert
Hit Rate 0%
Dodge 0%
AUTO
Lv. 6
warlock01
10200 / 10200
HP
Cancel
TURN OFF AUTO MODE
TOP TACTICS | SRPG
Android & IOS
2018

# Game of Life

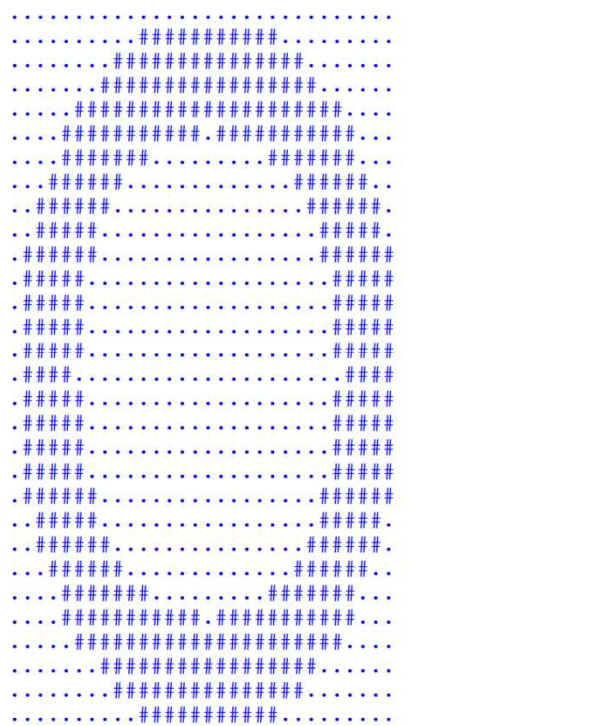
- https://bitstorm.org/gameoflife/



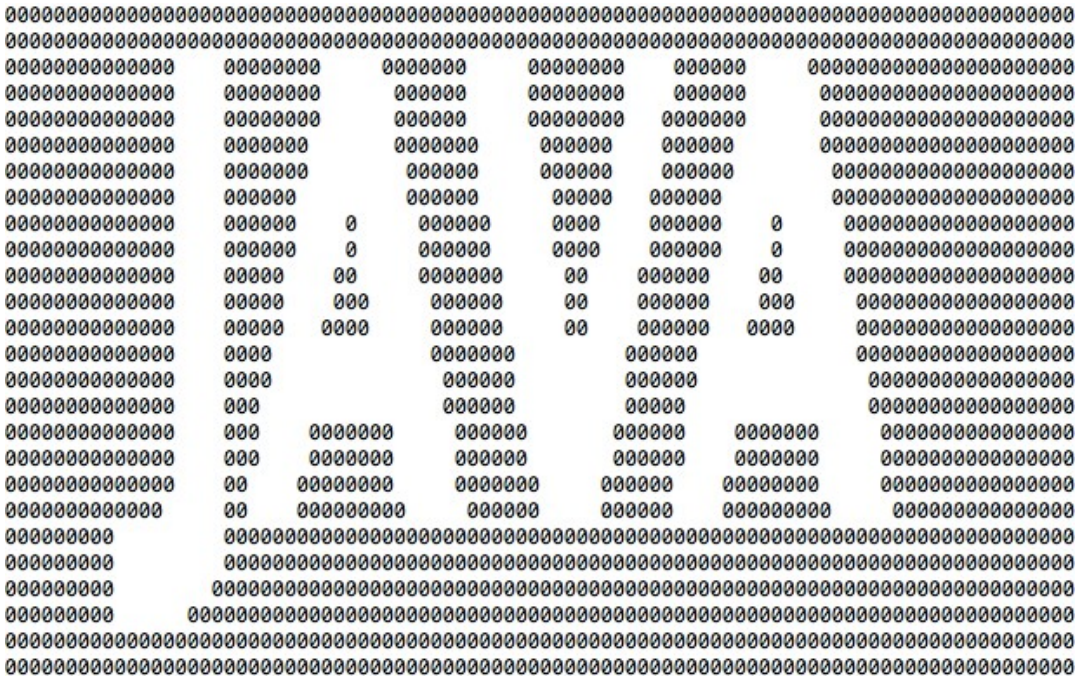
- [Virus Riddle]

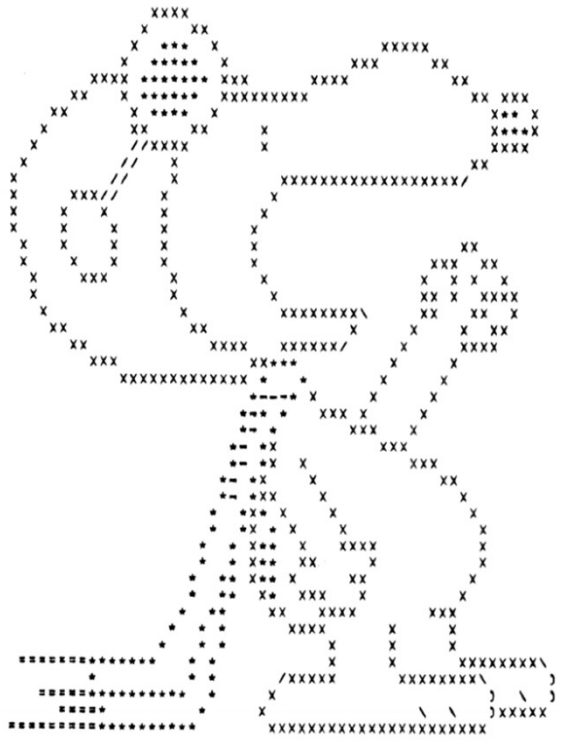
# circlePic()

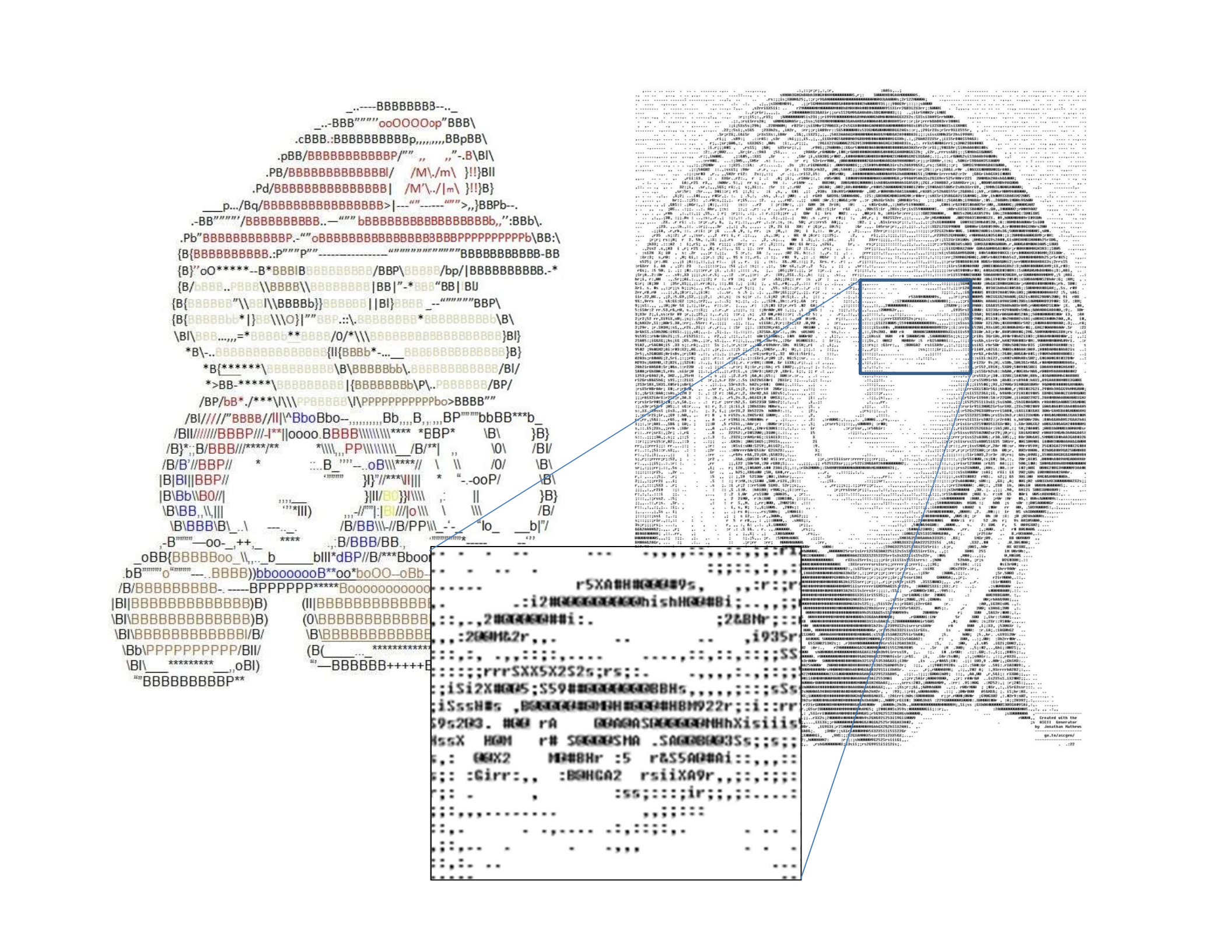
```
def circlePic():
    l = 30
    center = l/2
    pic = createZeroMatrix(l,l)
    for i in range(l):
        for j in range(l):
            if (l/3)**2 < (i-center)**2 + (j-center)**2 < (l/2)**2:
                pic[i][j] = "#"
            else:
                pic[i][j] = "."
    mTightPrint(pic)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

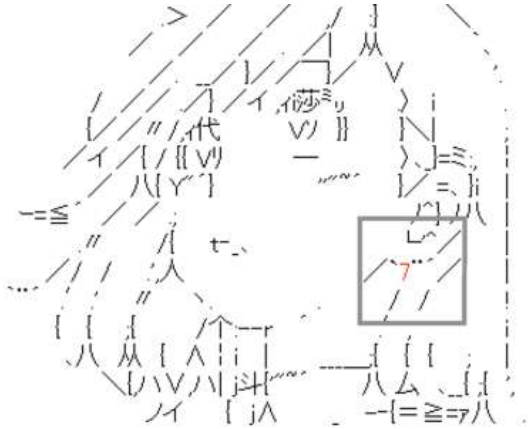


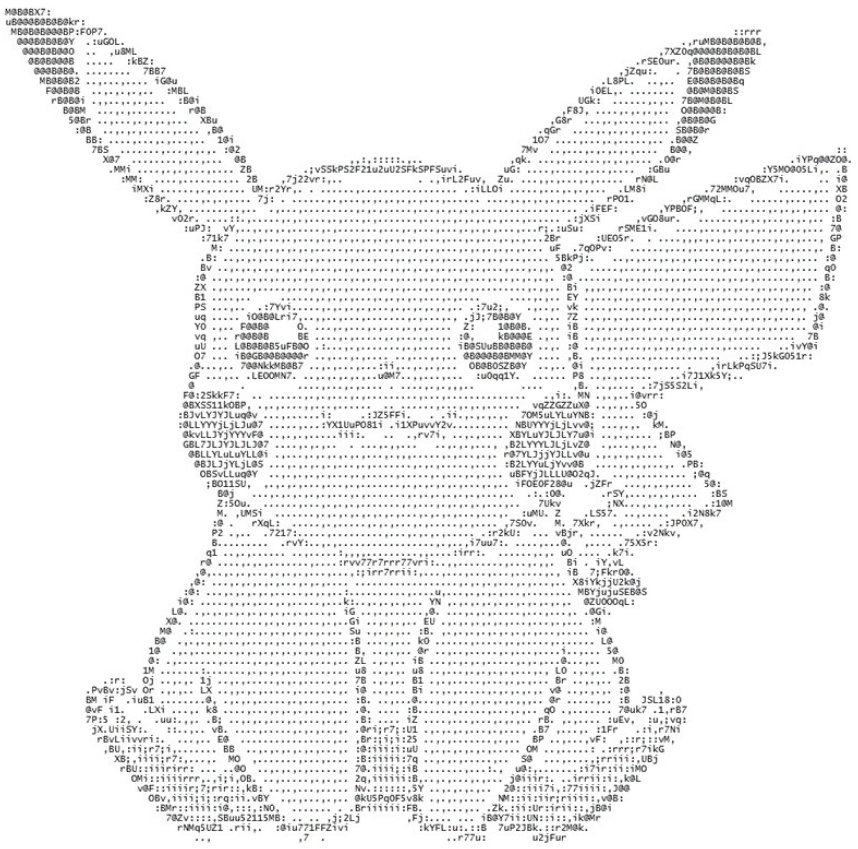
# ASCII Arts

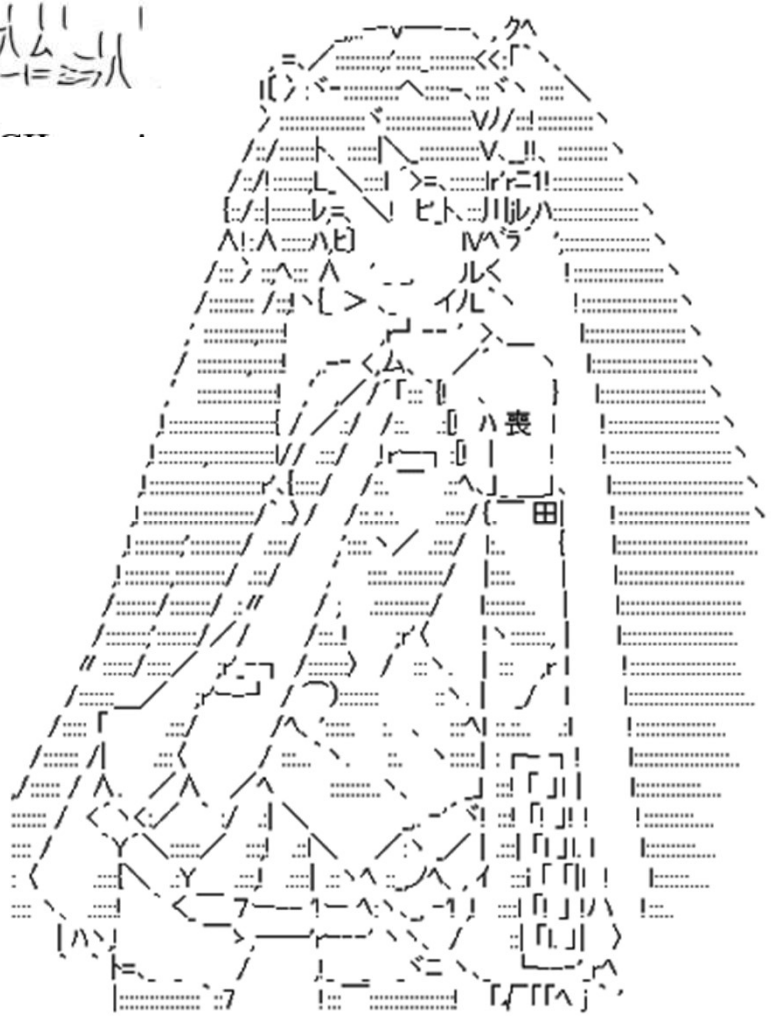





Created with the
ASCII Generator
by Jonathan Mathews
go.to/ascgen/

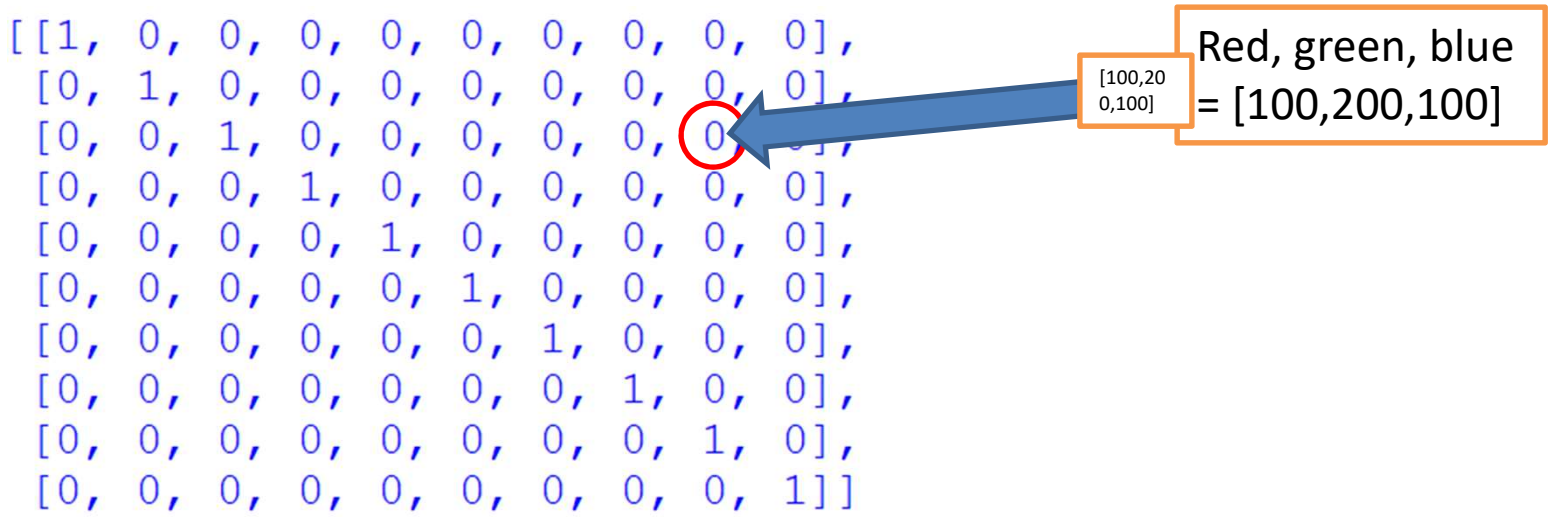






# Wait…

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



# 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

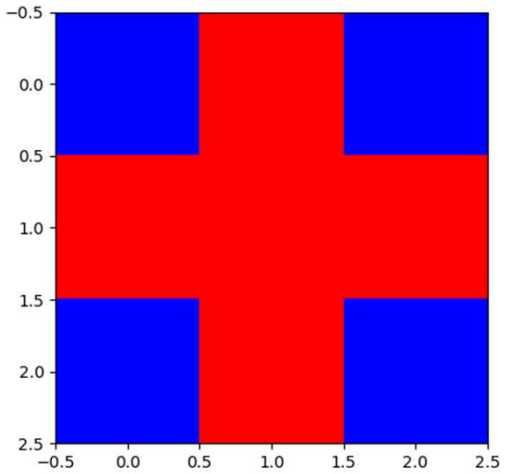
```
pic = [[[0,0,255],[255,0,0],[0,0,255]],
       [[255,0,0],[255,0,0],[255,0,0]],
       [[0,0,255],[255,0,0],[0,0,255]]]
```

One pixel is a list of three values of red, blue and green.
Usually ranging from 0 to 255

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

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

- 3 x 3 (x 3 for colors) pixels (zoomed)

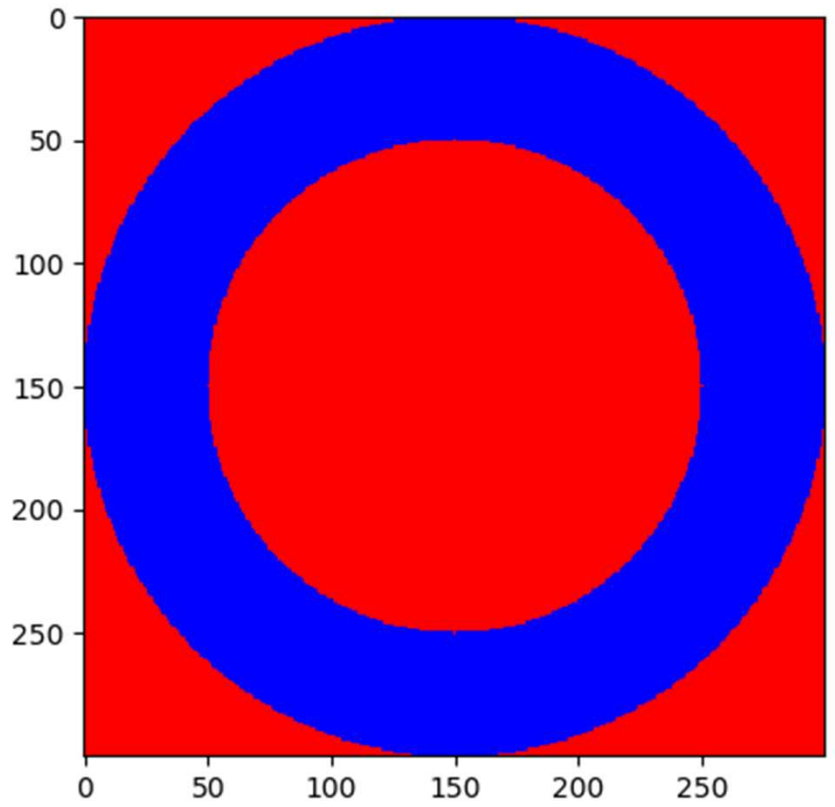


# Recap: circlePic()

```
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)
```

```
..............................
..........##########..........
........##############........
.......################.......
.....####################.....
....##########.##########....
....#######........#######....
...######..............######...
..######................######..
..#####..................#####..
.######................######
.#####..................#####
.#####..................#####
.#####..................#####
.#####..................#####
.####....................####
.#####..................#####
.#####..................#####
.#####..................#####
.#####..................#####
.######................######
..#####..................#####.
..######................######.
...######..............######..
....#######........#######....
....##########.##########....
.....####################.....
.......################.......
........##############........
..........##########..........
```

```
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()
```

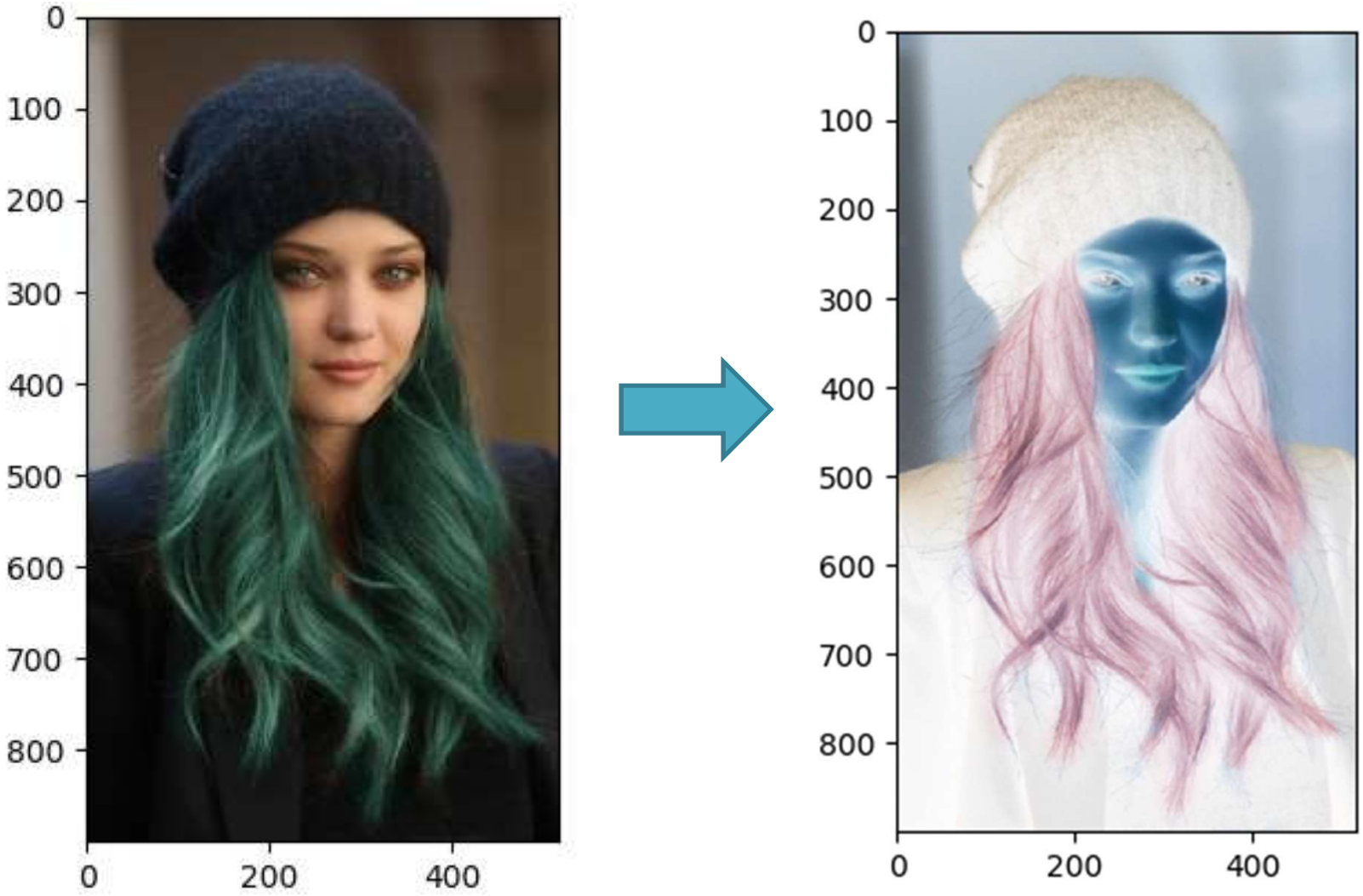


# Image Processing

Using the `imagio` 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()

    h = len(pic)
    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()
```

The picture has h x w numbers of pixels
(h = #rows, w = #columns)

Loop through each pixel with indices `(i,j)`

# 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

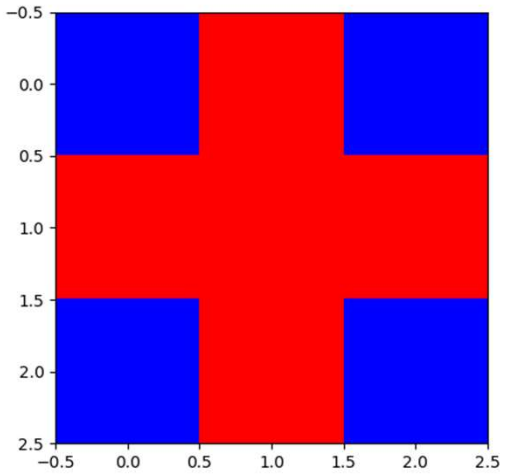
```
pic = [[[0,0,255],[255,0,0],[0,0,255]],
       [[255,0,0],[255,0,0],[255,0,0]],
       [[0,0,255],[255,0,0],[0,0,255]]]
```

One pixel is a list of three values of red, blue and green.
Usually ranging from 0 to 255

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

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

- 3 x 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] = ‘Smith’

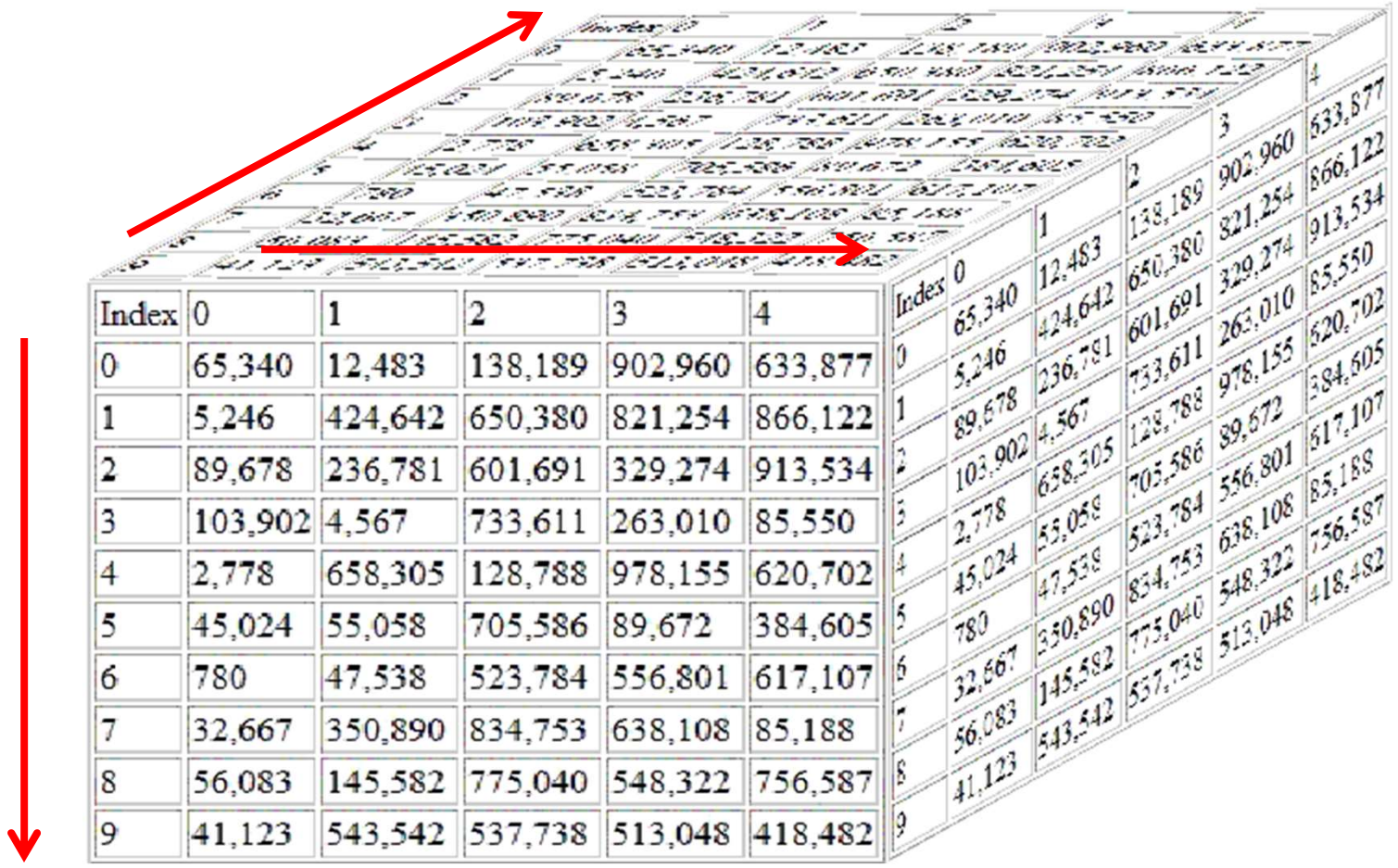
Two Dimensional Array, M

| | 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’

# Multi-Dimensional Array

- A three dimensional array



| Index | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 65,340 | 12,483 | 138,189 | 902,960 | 633,877 |
| 1 | 5,246 | 424,642 | 650,380 | 821,254 | 866,122 |
| 2 | 89,678 | 236,781 | 601,691 | 329,274 | 913,534 |
| 3 | 103,902 | 4,567 | 733,611 | 263,010 | 85,550 |
| 4 | 2,778 | 658,305 | 128,788 | 978,155 | 620,702 |
| 5 | 45,024 | 55,058 | 705,586 | 89,672 | 384,605 |
| 6 | 780 | 47,538 | 523,784 | 556,801 | 617,107 |
| 7 | 32,667 | 350,890 | 834,753 | 638,108 | 85,188 |
| 8 | 56,083 | 145,582 | 775,040 | 548,322 | 756,587 |
| 9 | 41,123 | 543,542 | 537,738 | 513,048 | 418,482 |

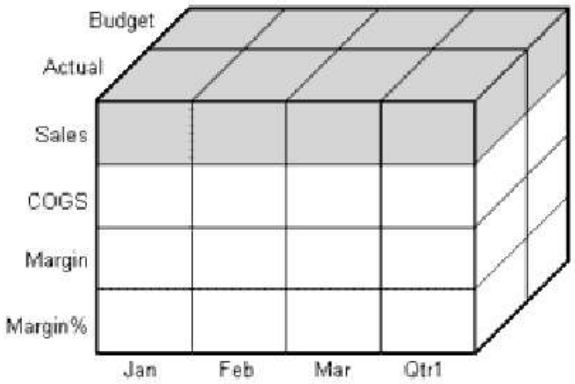
# Fancy Terminology in Business

- Multidimensional Data Model
- Multidimensional Analysis

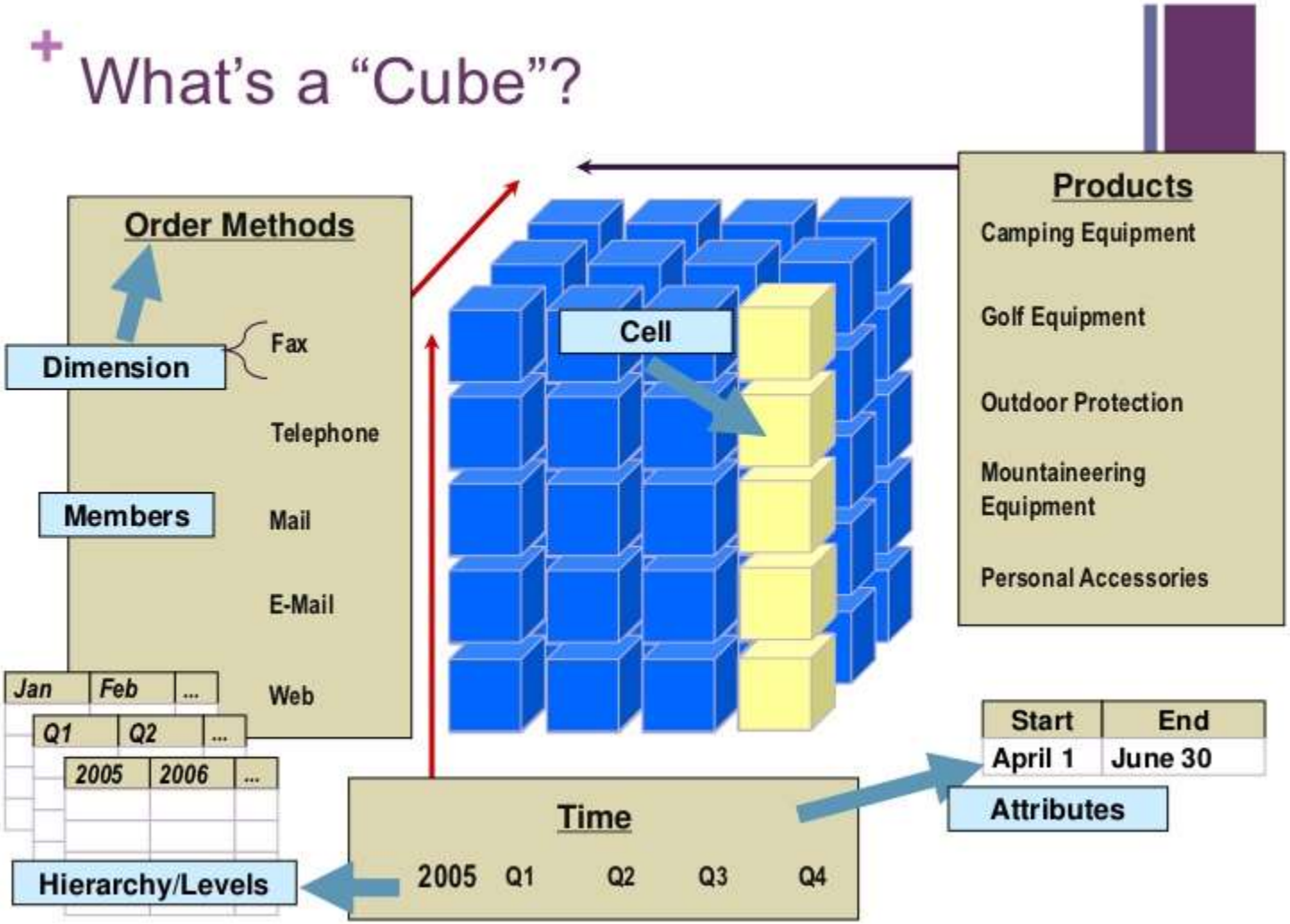
**Multidimensional Viewing and Analysis**

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.

*Sales Slice of the Database*

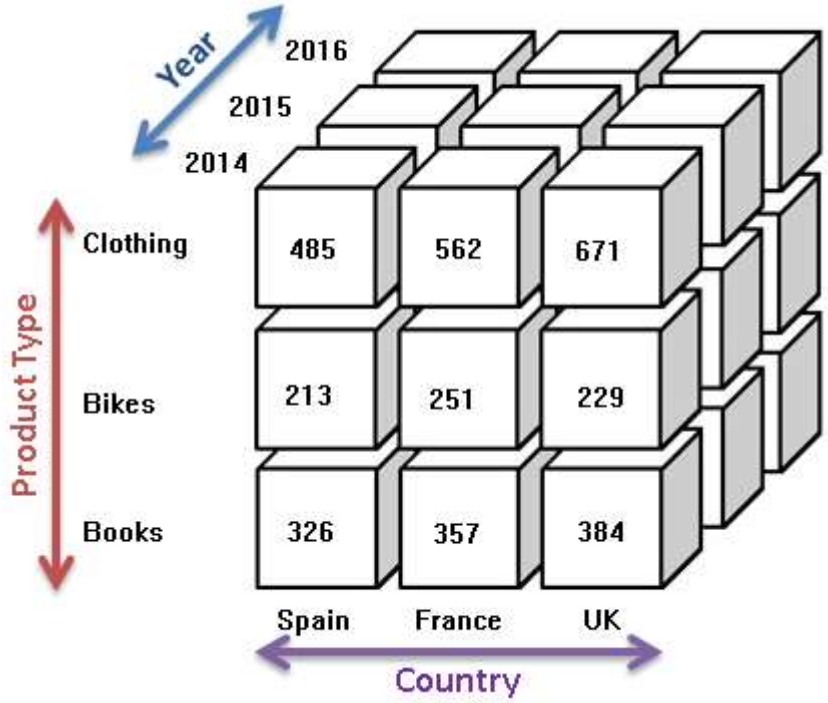


# What's a "Cube"?


Order Methods
Dimension
Fax
Telephone
Members
Mail
E-Mail
Web
Jan
Feb
...
Q1
Q2
...
2005
2006
...
Hierarchy/Levels
Cell
Products
Camping Equipment
Golf Equipment
Outdoor Protection
Mountaineering Equipment
Personal Accessories
Start
End
April 1
June 30
Attributes
Time
2005
Q1
Q2
Q3
Q4

# Sales Cube

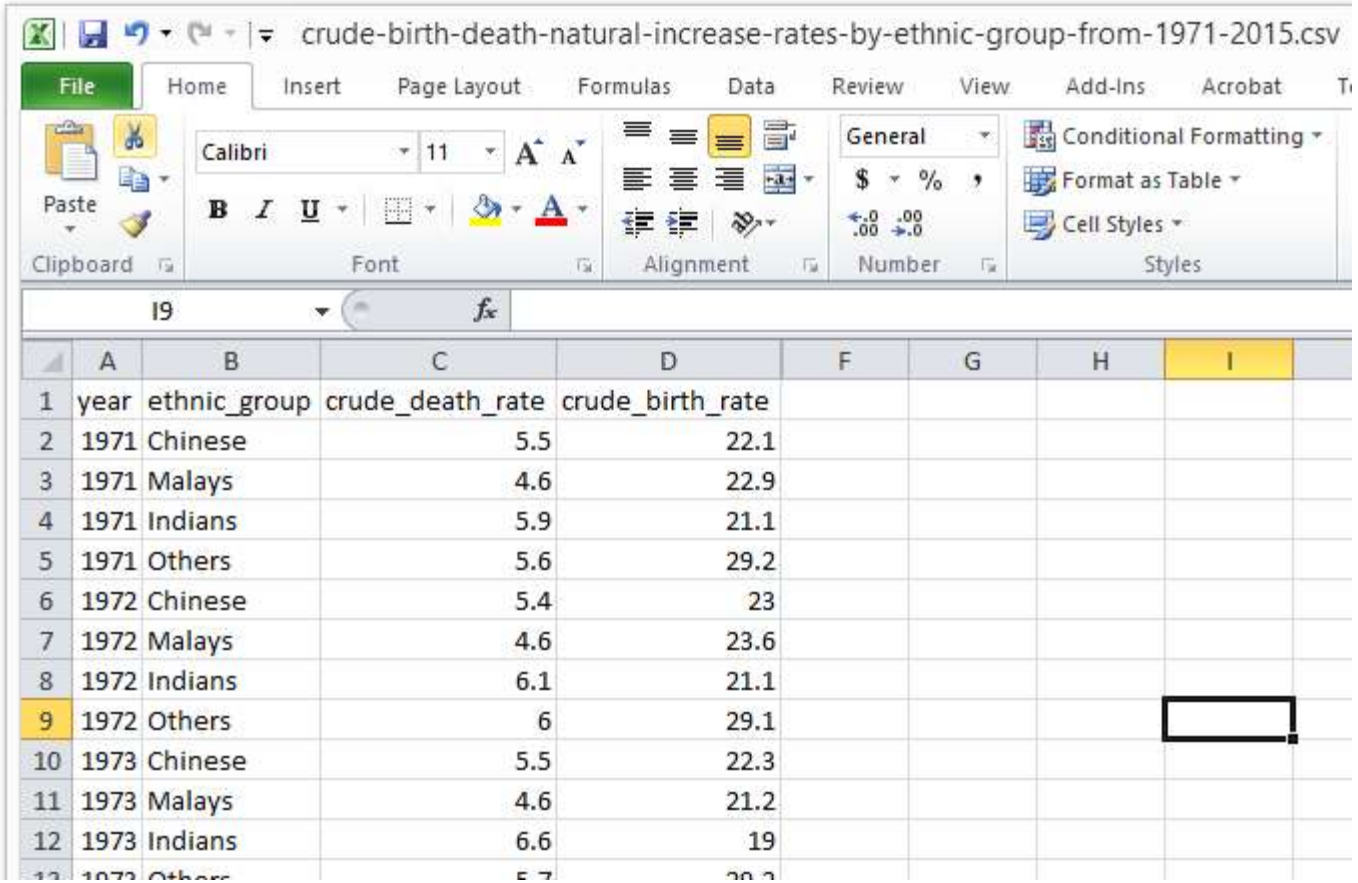
- Product Type x Year x Country




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

# Multi-Dimensional Array

- Year x Ethnic Group x {death/birth}

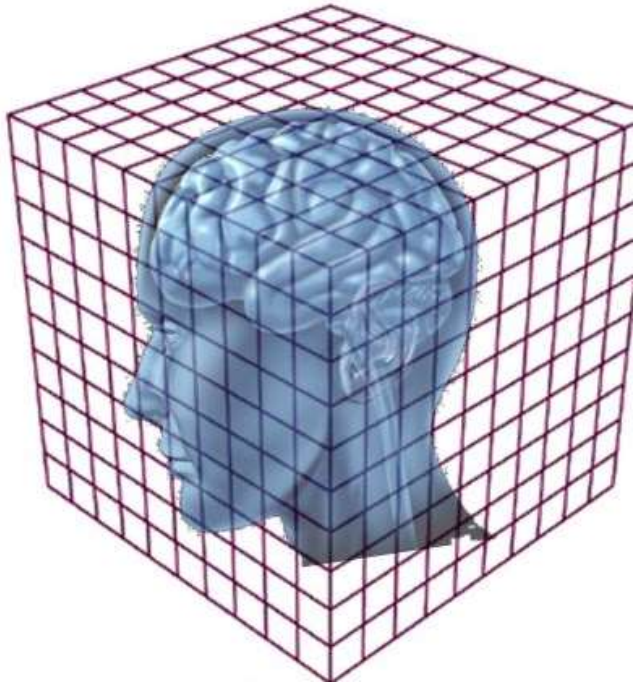


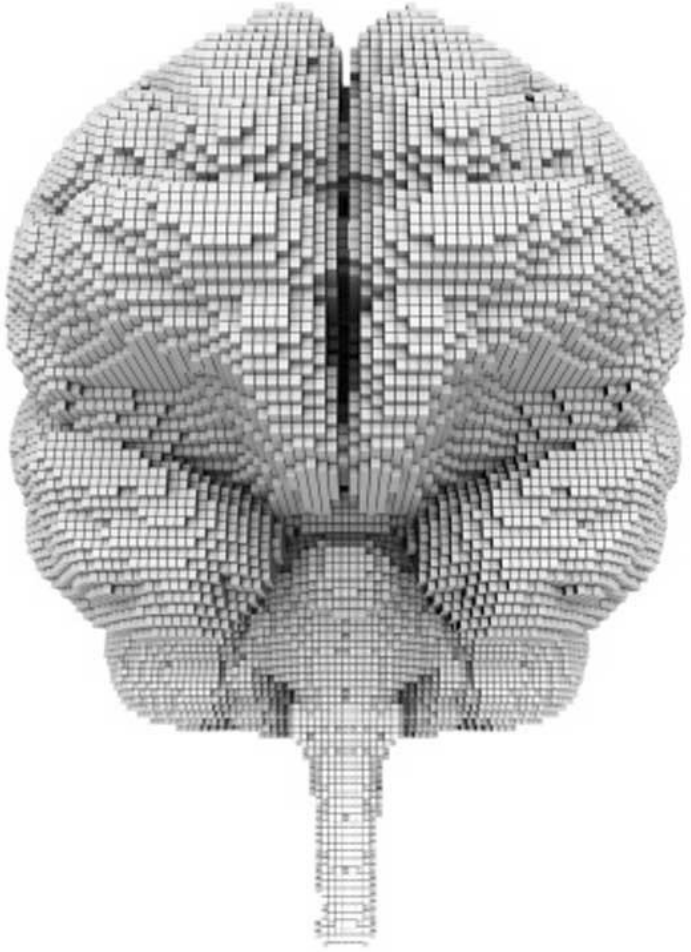

| | A | B | C | D | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| 1 | year | ethnic_group | crude_death_rate | crude_birth_rate | | | | |
| 2 | 1971 | Chinese | 5.5 | 22.1 | | | | |
| 3 | 1971 | Malays | 4.6 | 22.9 | | | | |
| 4 | 1971 | Indians | 5.9 | 21.1 | | | | |
| 5 | 1971 | Others | 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 | | | | |

# And…. how can we forget?!

# Voxels

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

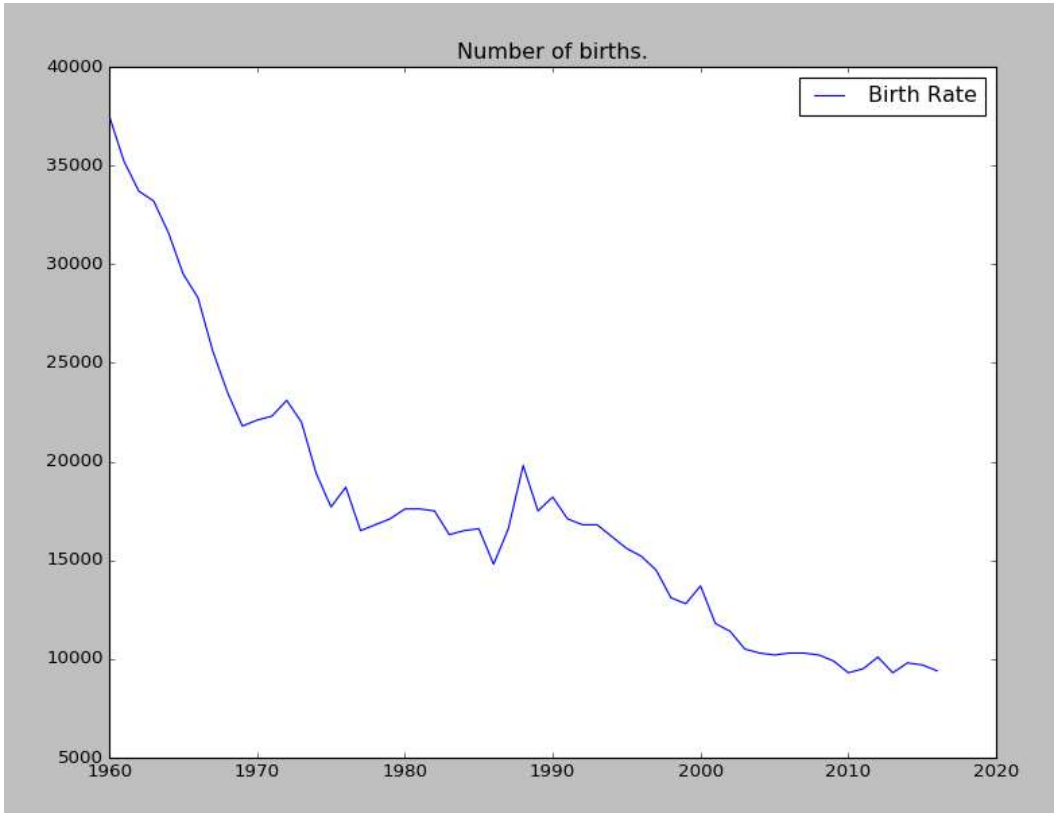




# Reading a CSV File

# 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.


Are you pulling my leg?
BRAINLESS TALES.com

# 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,