Multi-dimensional Array

Before That, Let's Introduce Numpy

- A convenient package
- You can get around with "normal" Python method but Numpy can shorten your code a lot

- For example, if you want to create a list of numbers from 0 to π without Numpy you have to:

```
x = [i/100 for i in range(0,314)]
```

With Numpy

```
>>> import numpy as np
>>> x1 = np.arange(0, 3.14, 0.1)
>>> x1
array([ 0. ,  0.1,  0.2,  0.3,  0.4,
  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,
        1.1,  1.2,  1.3,  1.4,  1.5,
  1.6,  1.7,  1.8,  1.9,  2. ,  2.1,
        2.2,  2.3,  2.4,  2.5,  2.6,
  2.7,  2.8,  2.9,  3. ,  3.1])
>>> len(x1)
32
```

Be Careful!!!
This is not “arrange”
but “arange”

Another Way

Be Careful!!!
This is not
“linespace” but
“linspace”

```
>>> x2 = np.linspace(0,3.14,100)
>>> x2
array([ 0.        ,  0.03171717,  0.063
43434,  0.09515152,  0.12686869,
        0.15858586,  0.19030303,  0.222
0202 ,  0.25373737,  0.28545455,
        0.31717172,  0.34888889,  0.380
                .
                .
                .
        2.6959596 ,  2.7276677,  2.759
39394,  2.79111111,  2.82282828,
        2.85454545,  2.88626263,  2.917
9798 ,  2.94969697,  2.98141414,
        3.01313131,  3.04484848,  3.076
56566,  3.10828283,  3.14      ])
>>> len(x2)
100
```

Remember Numpy?

- So far, we have learned that Numpy can create an array that is like a list

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> a
array([1, 2, 3])
>>> a[0]=999
>>> a
array([999,   2,   3])
>>> type(a)
<class 'numpy.ndarray'>
>>> l = [1,2,3]
>>> type(l)
<class 'list'>
```

Broadcasting

Array Broadcasting

```
>>> a = np.array([1,2,3,4,5])
>>> a + 1
array([2, 3, 4, 5, 6])
>>> a * 3
array([ 3,  6,  9, 12, 15])
>>> a > 5
array([False, False, False, False, False], dtype=bool)
```

"Broadcasting"

Different from LIST

Create another array with the Boolean results

Array Broadcasting

```
>>> a = np.array([1,2,3,4,5])
>>> a + 1
array([2, 3, 4, 5, 6])
>>> a * 3
array([ 3,  6,  9, 12, 15])
>>> a > 5
array([False, False, False, False, False], dtype=bool)
```

- A single operator with a single number at the right side
  - Apply that operation to every single entry of the array

Array Math

```
>>> v1 = np.array([1,2,3])
>>> v2 = np.array([4,9,16])
>>> print(v1+v2)
[ 5 11 19]
>>> print(v1*v2)
[ 4 18 48]
>>> np.sqrt(v2)
array([ 2.,  3.,  4.])

>>> v1.dot(v2)
70
>>> v2.sum()
29
```

Different from list !!!!

Directly apply on EVERY element in the array

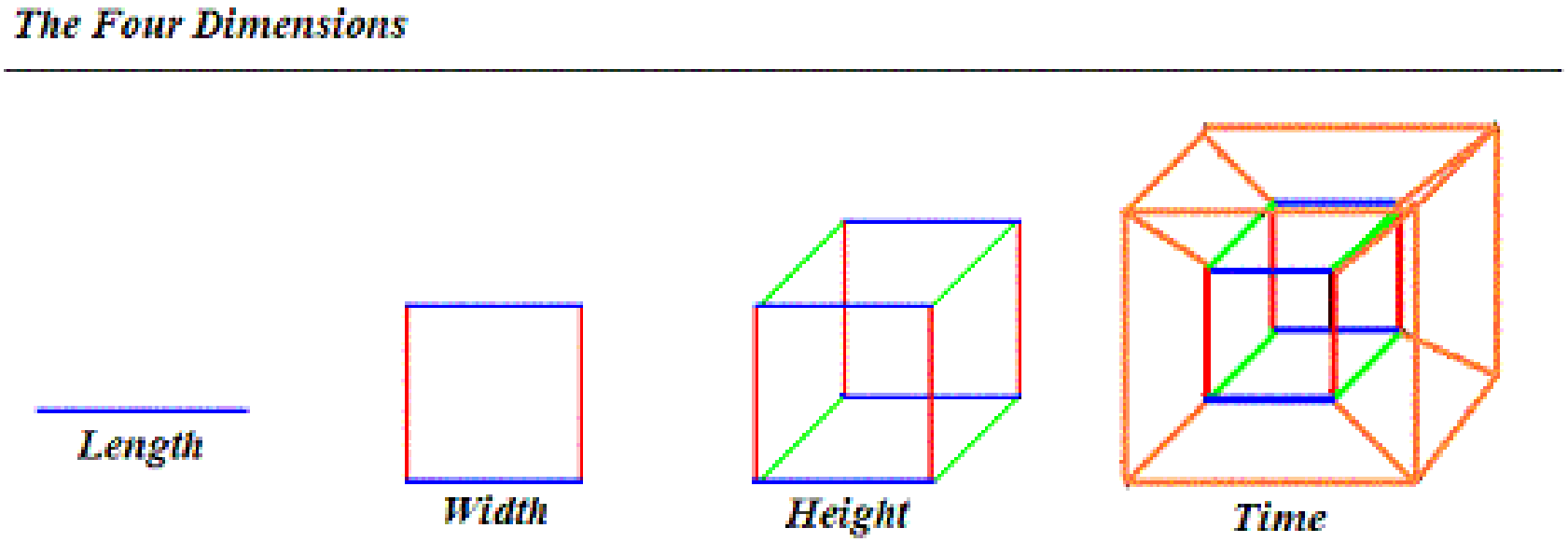
Dot product

More of these functions:
https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.math.html

Two-dimensional Array

Dimensions

- Are we living in a three dimensional space?



[The xkcd guide to the fourth dimension]

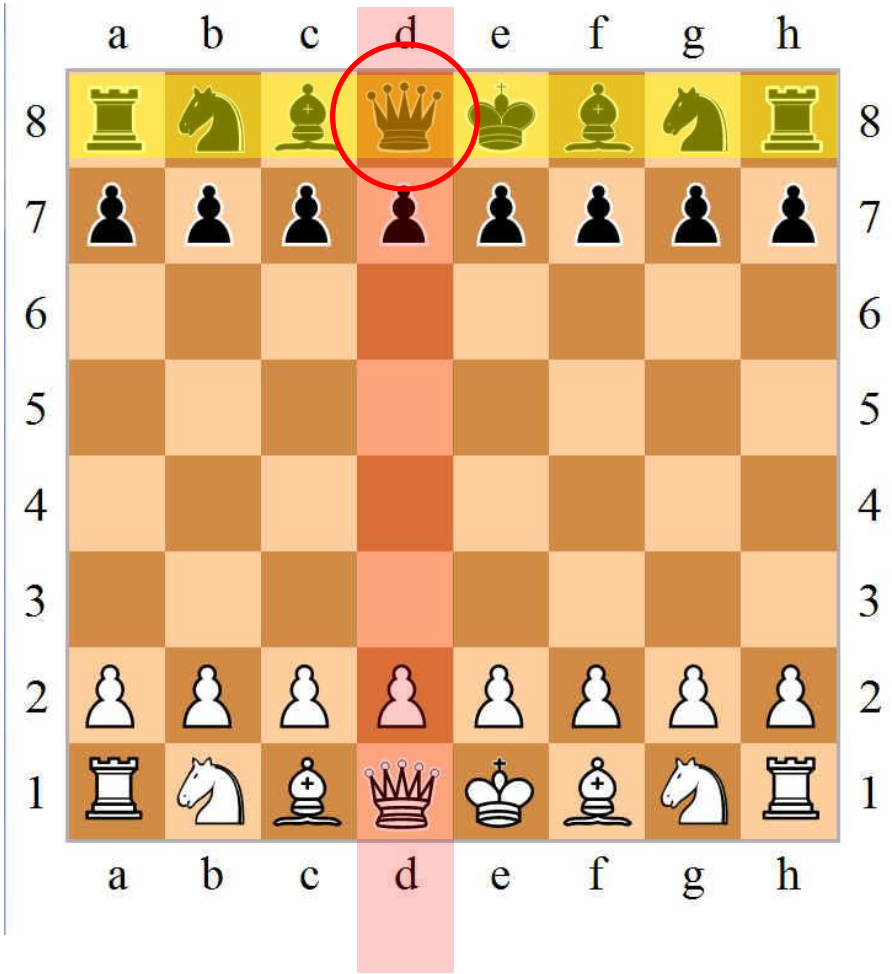
Motel in US

One Dimension

Hotel

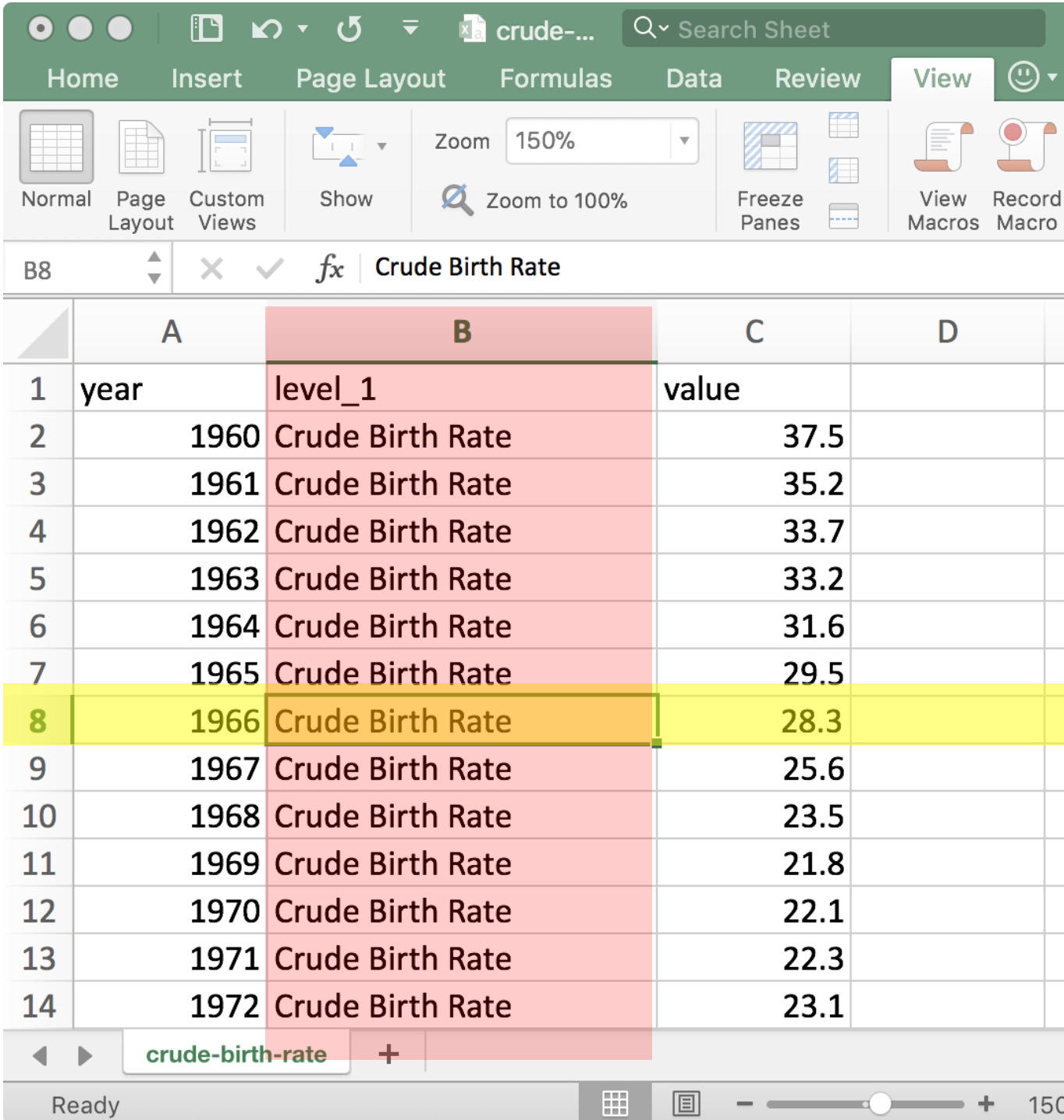
My room is 02-05

Chess Position



a8	b8	c8	d8	e8	f8	g8	h8
a7	b7	c7	d7	e7	f7	g7	h7
a6	b6	c6	d6	e6	f6	g6	h6
a5	b5	c5	d5	e5	f5	g5	h5
a4	b4	c4	d4	e4	f4	g4	h4
a3	b3	c3	d3	e3	f3	g3	h3
a2	b2	c2	d2	e2	f2	g2	h2
a1	b1	c1	d1	e1	f1	g1	h1

- 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

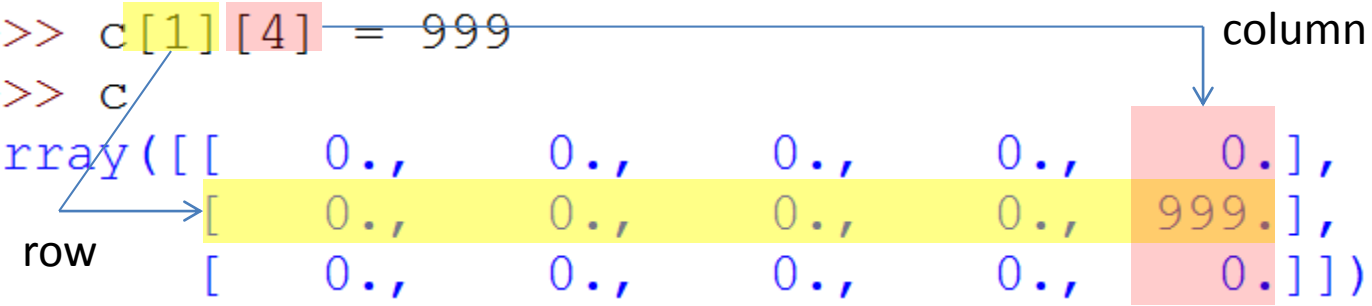
	0	1	2	3
0	‘Apple’	‘Lah’	‘Cat’	‘Eve’
1	‘Hello’	‘Pay’	‘TV’	‘Carl’
2	‘What’	‘Bank’	‘Radio’	‘Ada’
3	‘Frog’	‘Peter’	‘Sea’	‘Eat’
4	‘Job’	‘Fry’	‘Gym’	‘Wow’
5	‘Walk’	‘Fly’	‘Cook’	‘Look’

M[4][1] = ‘Fry’

Locating an Entry

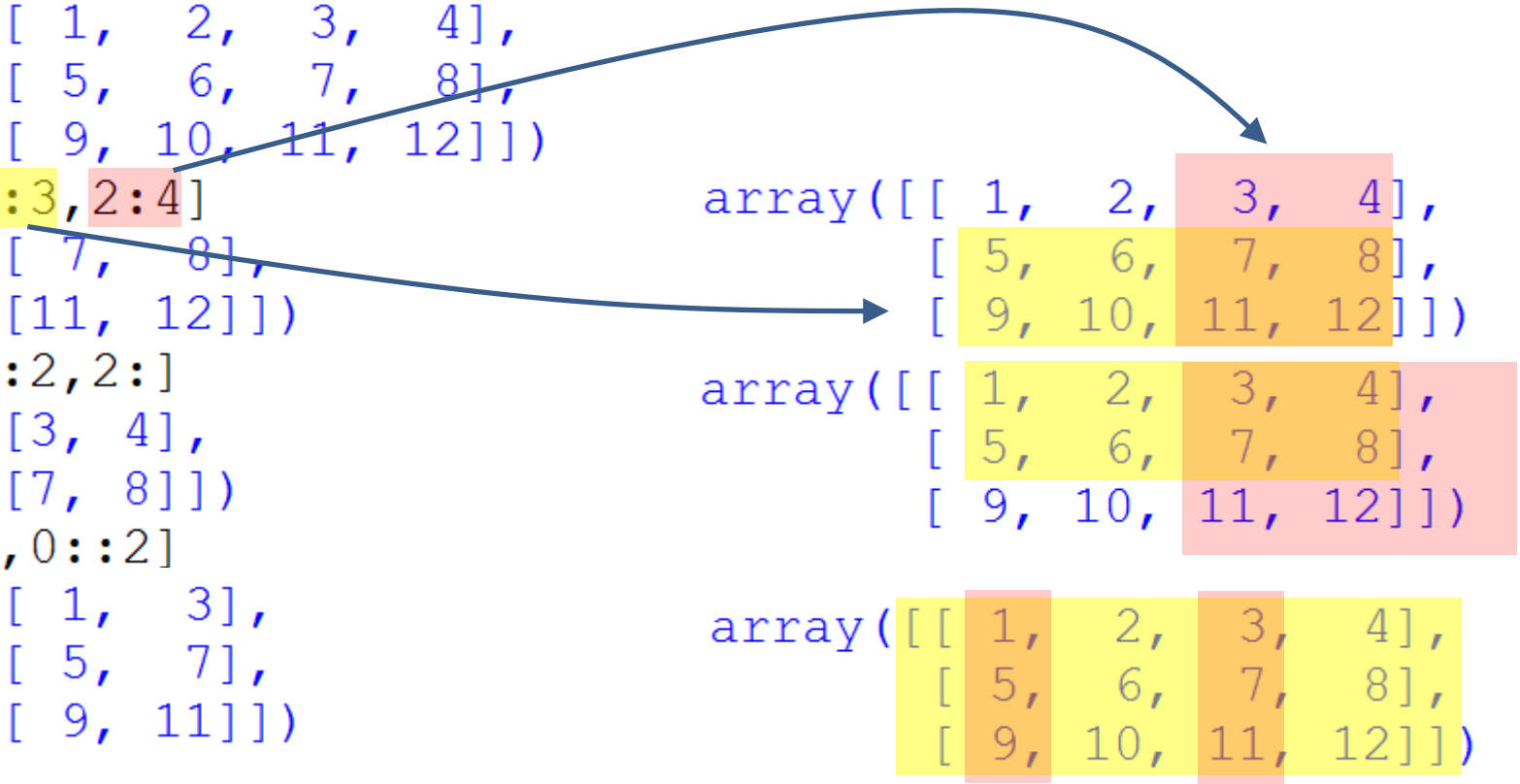
- Creating a 3x5 array with all zeros

```
>>> c = np.zeros((3,5))
>>> c
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
>>> c[1][4] = 999
>>> c
array([[   0.,    0.,    0.,    0.,    0.],
       [   0.,    0.,    0.,    0.,  999.],
       [   0.,    0.,    0.,    0.,    0.]])

>>> c[1][4]
999.0
>>> c[1,4]
999.0
```



Sub-array (Sub-matrix)

```
>>> a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
>>> a
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
>>> a[1:3,2:4]
array([[ 7,  8],
       [11, 12]])
>>> a[0:2,2:]
array([[3, 4],
       [7, 8]])
>>> a[:,0::2]
array([[ 1,  3],
       [ 5,  7],
       [ 9, 11]])
```



Matrix Multiplication

```
>>> m1 = np.array([[2,1],[3,10]])
>>> m2 = np.array([[4,0],[0,3]])
>>> m1
array([[ 2,  1],
       [ 3, 10]])
>>> m2
array([[4, 0],
       [0, 3]])
>>> np.matmul(m1,m2)
array([[ 8,  3],
       [12, 30]])
>>> m3 = np.array([[1,2,3,4,5]])
>>> np.matmul(m1,m3)
Traceback (most recent call last):
  File "<pyshell#119>", line 1, in <module>
    np.matmul(m1,m3)
ValueError: shapes (2,2) and (1,5) not aligned: 2 (dim 1)
!= 1 (dim 0)
```

Boolean Array Indexing

```
>>> a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
>>> a
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])

>>> bool_idx = (a > 5)
>>> print(bool_idx)
[[False False False False]
 [False  True  True  True]
 [ True  True  True  True]]
>>> print(a[bool_idx])
[ 6  7  8  9 10 11 12]
>>> print(a[a>10])
[11 12]
```

Multi-dimensional Array

Dimensions

One Dimensional Array, A

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

A[5] = ‘Smith’

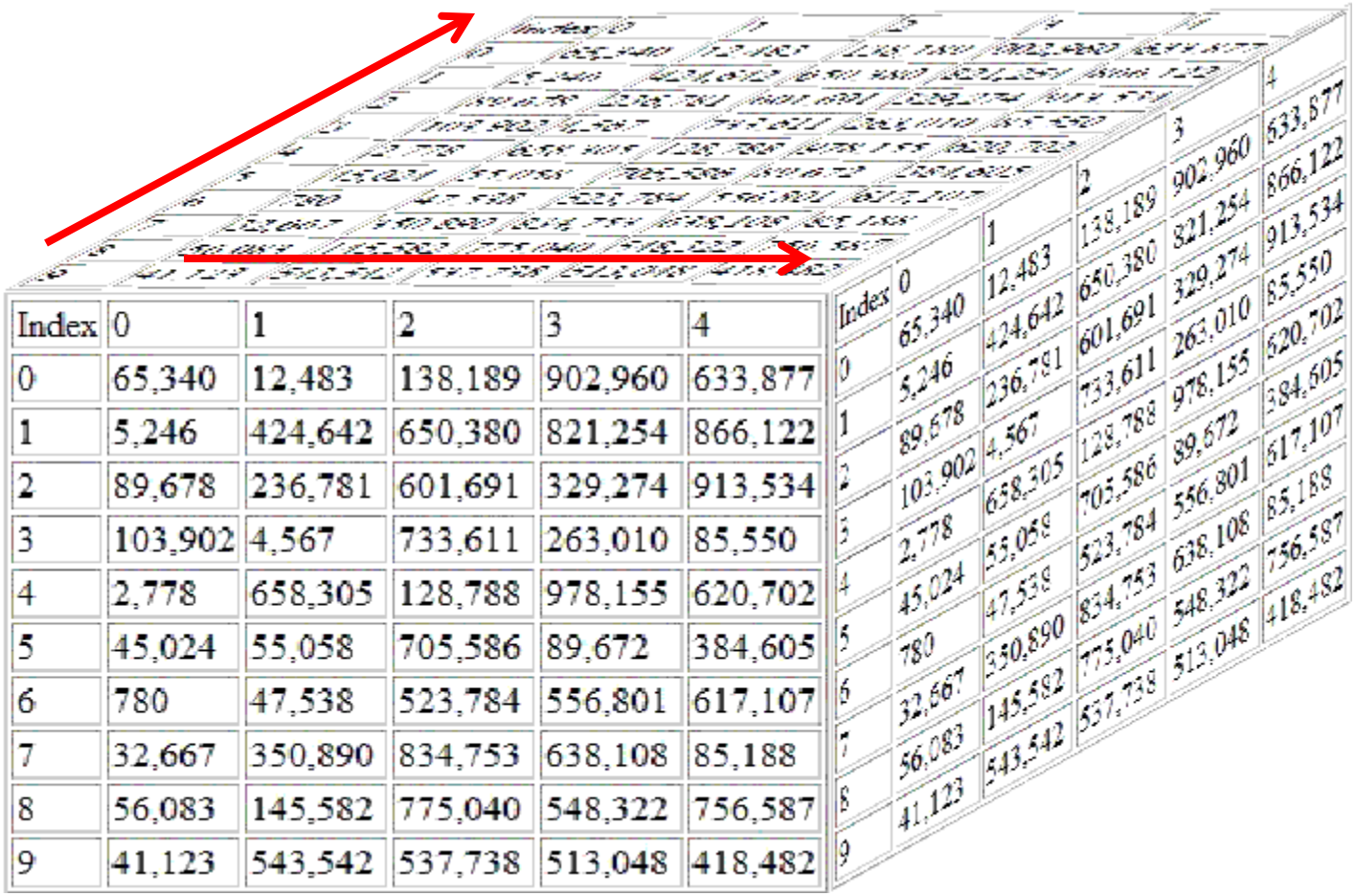
Two Dimensional Array, M

	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

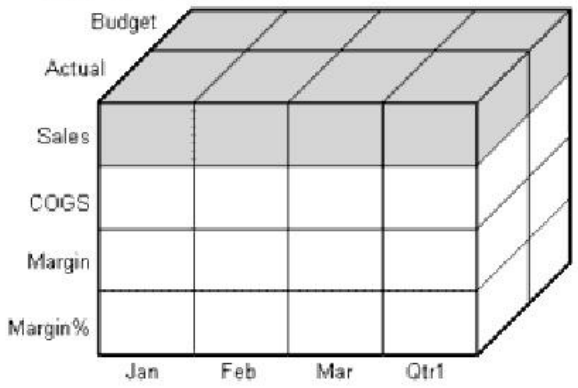
Fancy Terminology in Business

- Multidimensional Data Model
- Multidimensional Analysis

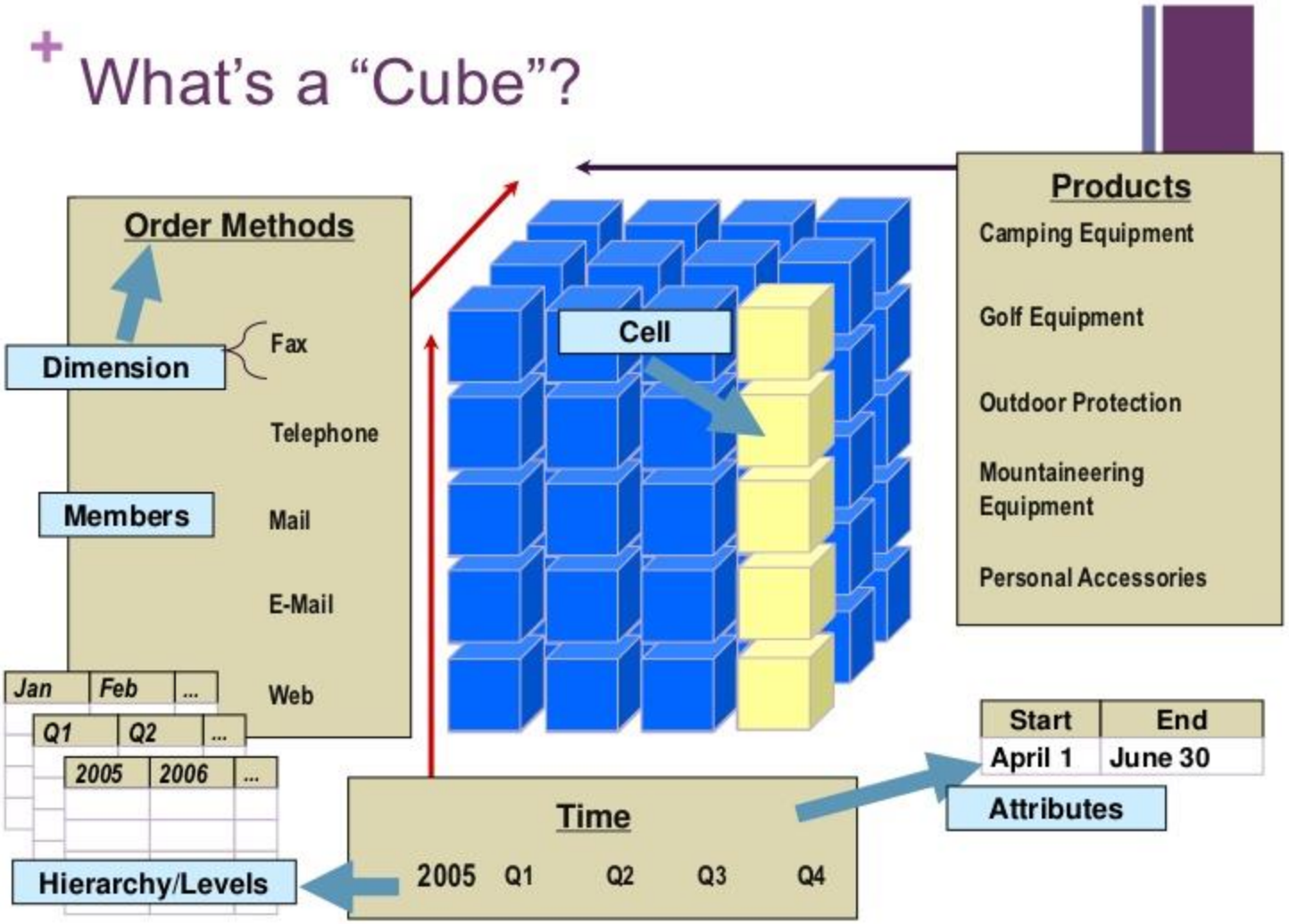
Multidimensional Viewing and Analysis

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

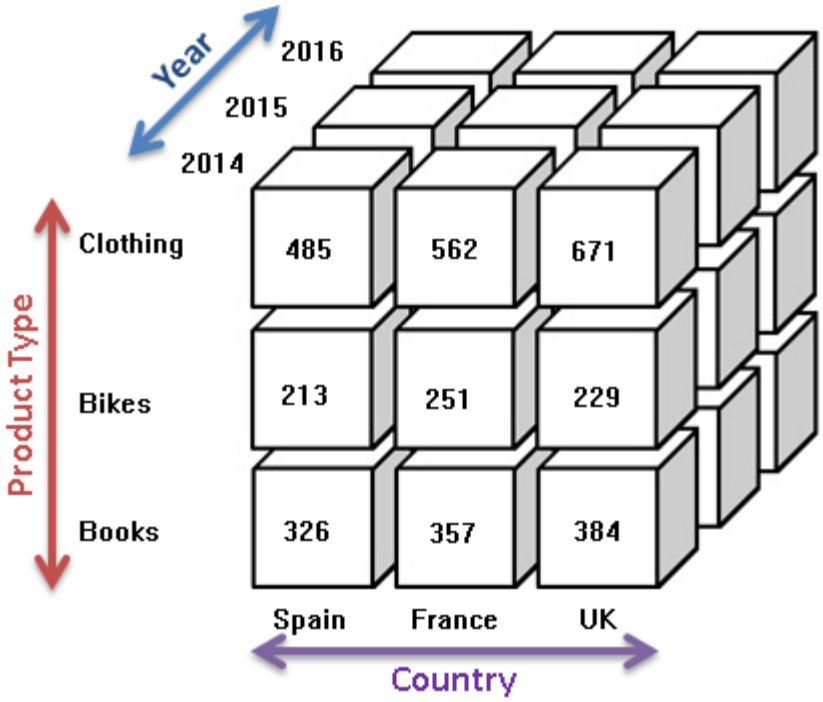


What's a "Cube"?


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

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

- A three dimensional array
  - E.g. a picture:
    - N x M pixels, and each pixel is an array of three colors, (R, G, B) (Dimensions = N x M x 3)
  - Next Week:

```
from scipy import misc,ndimage
import matplotlib.pyplot as plt

cat_pic = misc.imread('cute cat.jpg')
cat_pic2 = 255 - cat_pic
plt.figure(1)
plt.imshow(cat_pic)
plt.figure(2)
plt.imshow(cat_pic2)
plt.show()
```

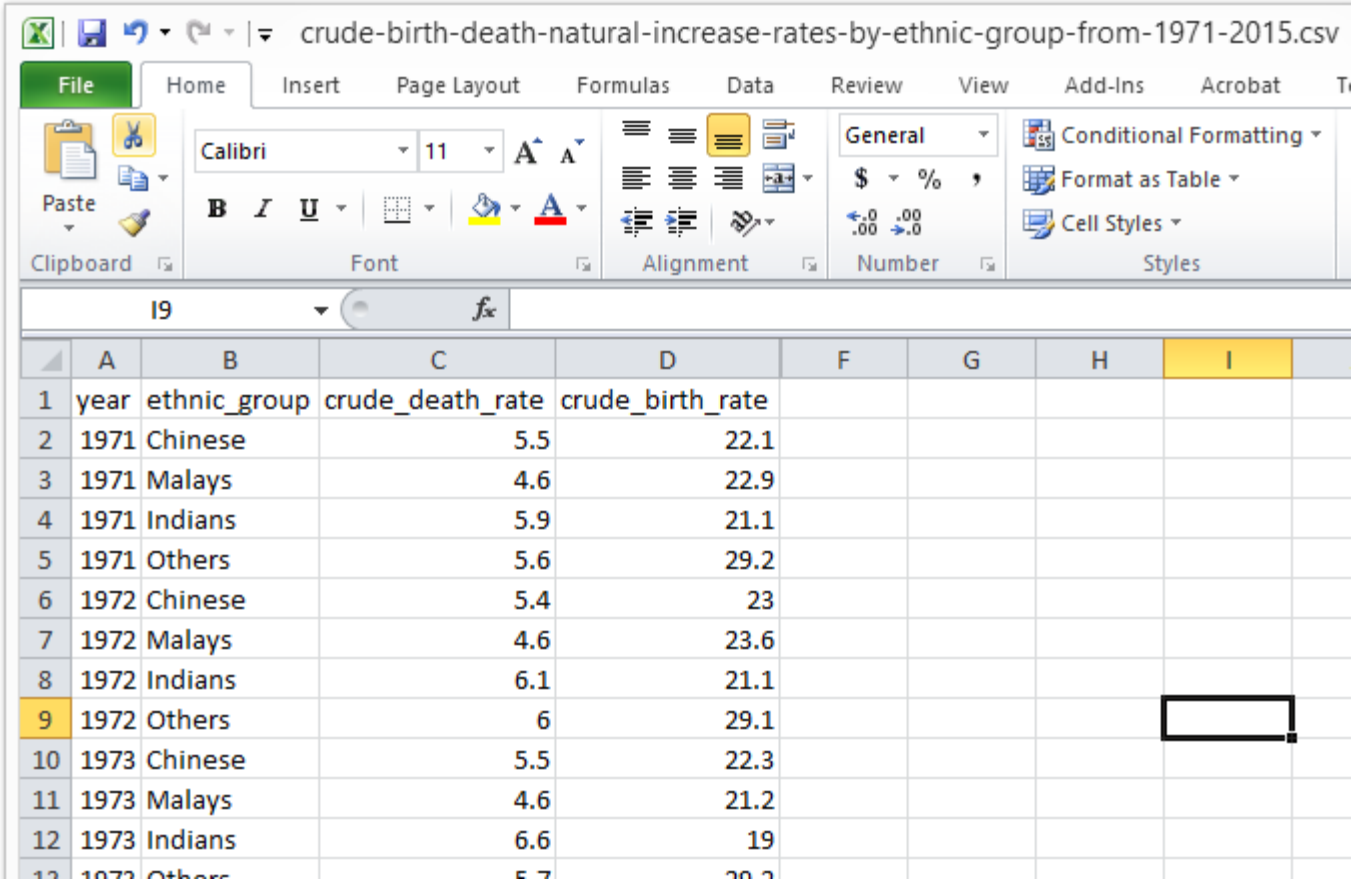
Image Processing

- Original Picture

- Negative Image

Multi-Dimensional Array

- Year x Ethnic Group x {death/birth}



crude-birth-death-natural-increase-rates-by-ethnic-group-from-1971-2015.csv

File Home Insert Page Layout Formulas Data Review View Add-Ins Acrobat

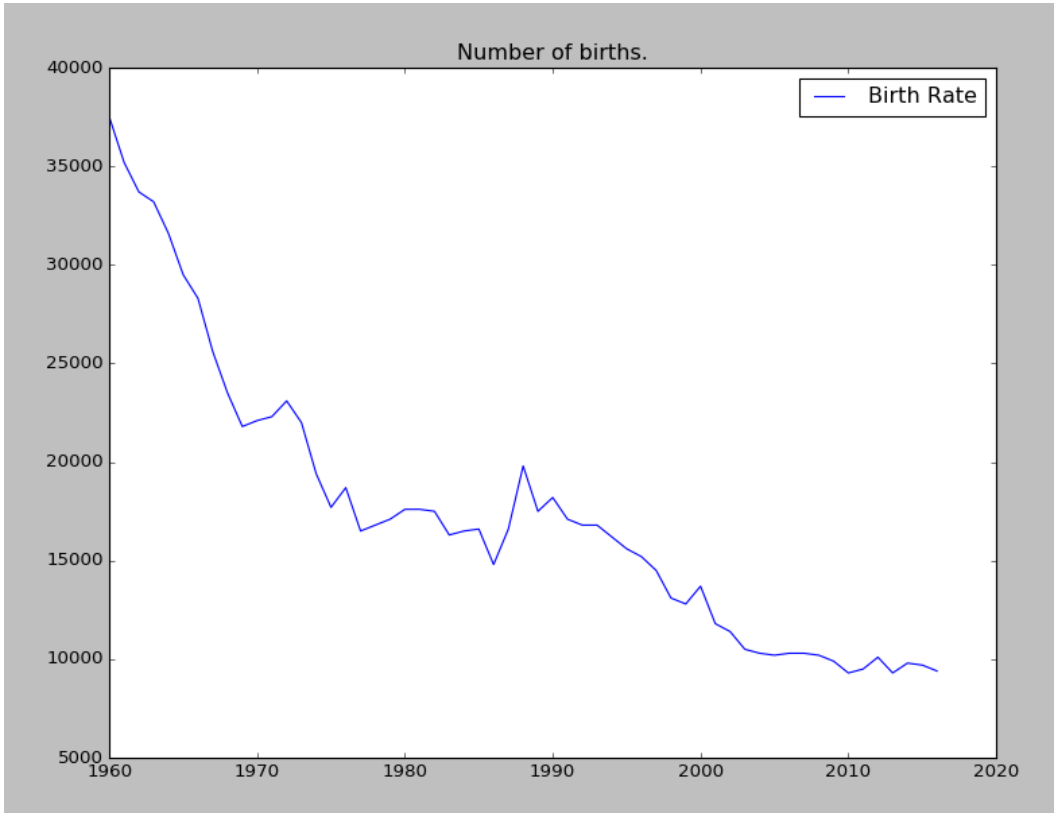
Paste Clipboard Calibri 11 Font Alignment General Number Conditional Formatting Format as Table Cell Styles Styles

I9

	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				

2D Array x CSV

- Remember last week, 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()
```


Are you pulling my leg?
BRAINLESS TALES.com

Reading CSV File

Create a CSV File Reader

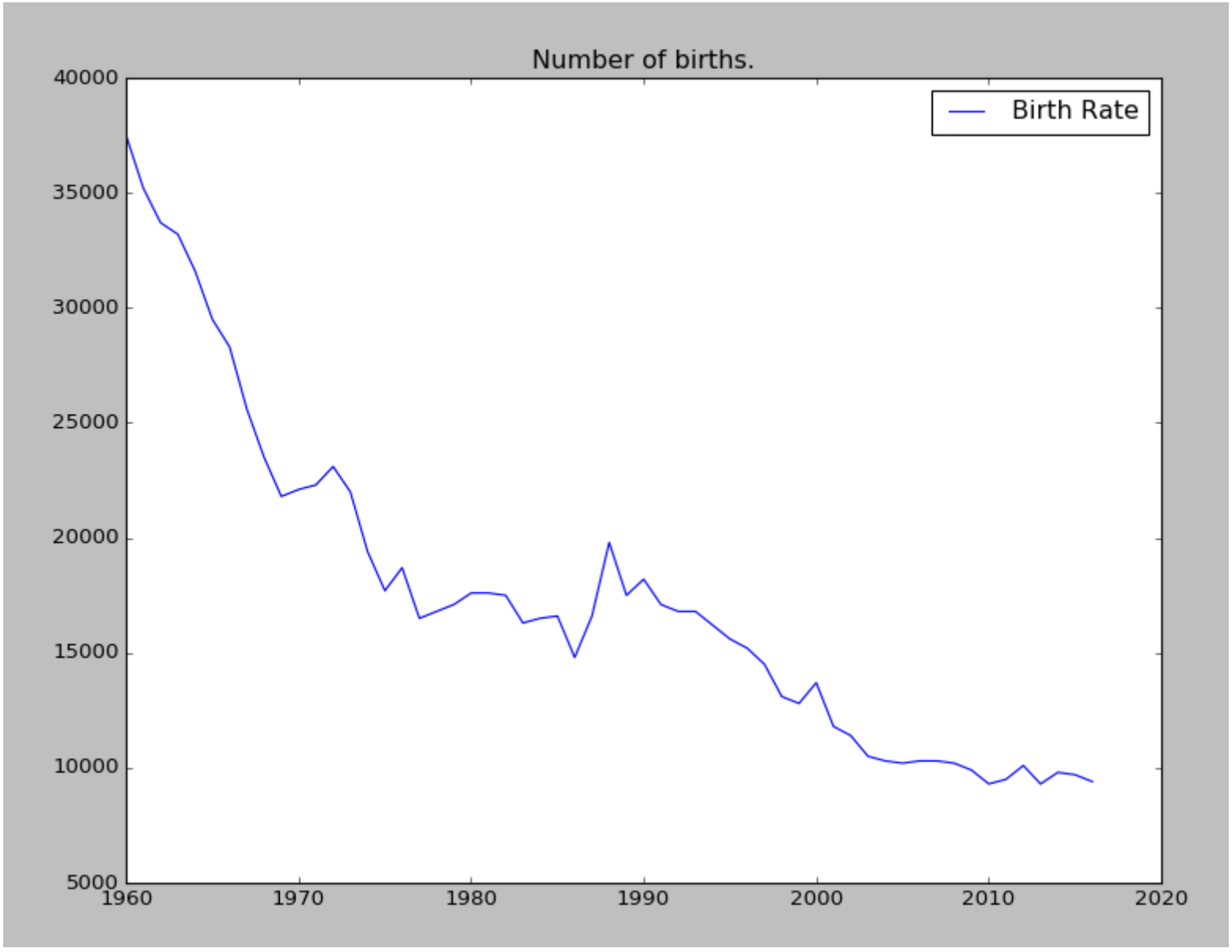
Read in a CSV file into a list

```
>>> import csv
>>> birth_file = open('crude-birth-rate.csv')
>>> birth_file_reader = csv.reader(birth_file)
>>> data_in_list = list(birth_file_reader)
>>> print(data_in_list)
[['year', 'level_1', 'value'], ['1960', 'Crude Birth
Rate', '37.5'], ['1961', 'Crude Birth Rate', '35.2']
, ['1962', 'Crude Birth Rate', '33.7'], ['1963', 'Cr
ude 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', '
```

```
import matplotlib.pyplot as plt
import numpy as np
import csv

def plot_birth_rate():
    birth_file = open('crude-birth-rate.csv')
    birth_file_reader = csv.reader(birth_file)

    birth_data = np.array(list(birth_file_reader))
    year = birth_data[1:,0]
    num_birth = birth_data[1:,2]
    birth_file.close()

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


plot_birth_rate()
```

Because we want these cool Numpy features

Actually we should close the file always. But the "for" loop with file close it for you automatically


Number of births.
Birth Rate
40000
35000
30000
25000
20000
15000
10000
5000
1960
1970
1980
1990
2000
2010
2020