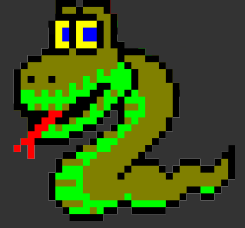


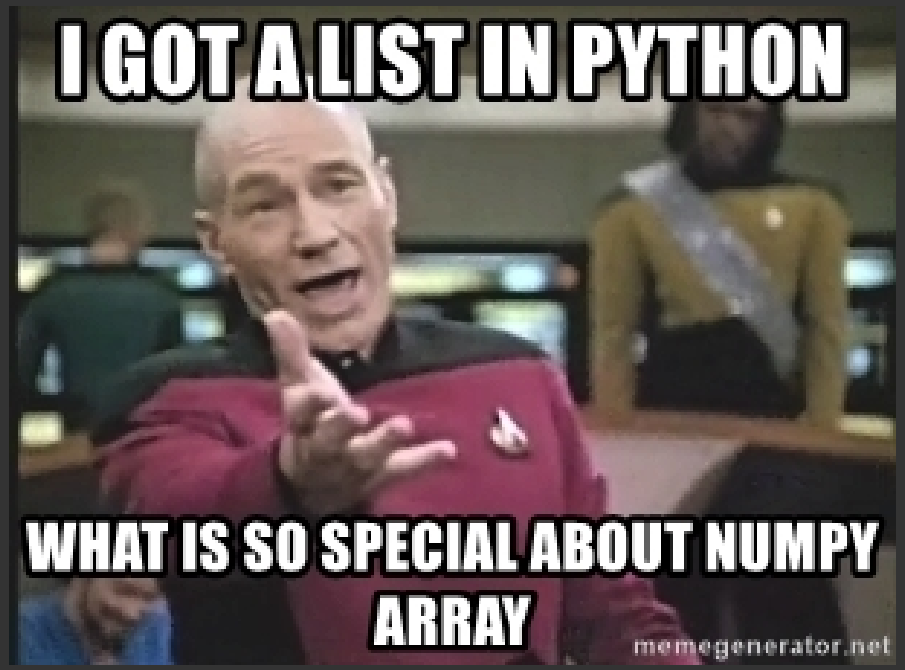
>>> Introduction to Data Science with Python
>>> DS101

Name: Celia Cintas[†] Nahuel Defosse [‡]

Date: February 23, 2019

[†]cintas.celia@gmail.com
[‡]nahuel.defosse@gmail.com

>>> What is numpy?

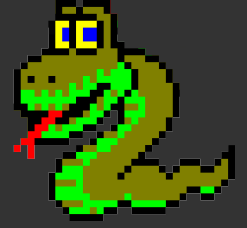


* extension package to Python for multi-dimensional arrays
* closer to hardware (efficiency)
* designed for scientific computation (convenience)
* Also known as array oriented computing

```
$ pip install numpy numpy-html
```

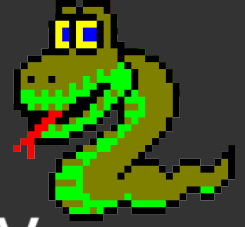
https://docs.scipy.org/doc/

>>> What is a matrix?



* A matrix is a collection of numbers arranged into a fixed number of rows and columns.
* A two dimensional matrix of 2x3 can be:
  $$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$
* Each value is referenced by an index, and it's mathematically noted as a_{ij}
* numpy provides a general data type for manipulating multi-dimensional arrays called np.array

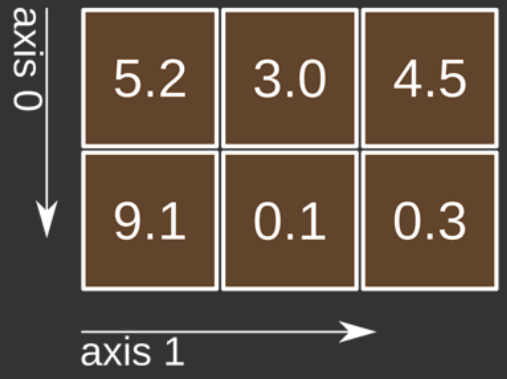
>>> 1D, 2D and 3D arrays



1D array

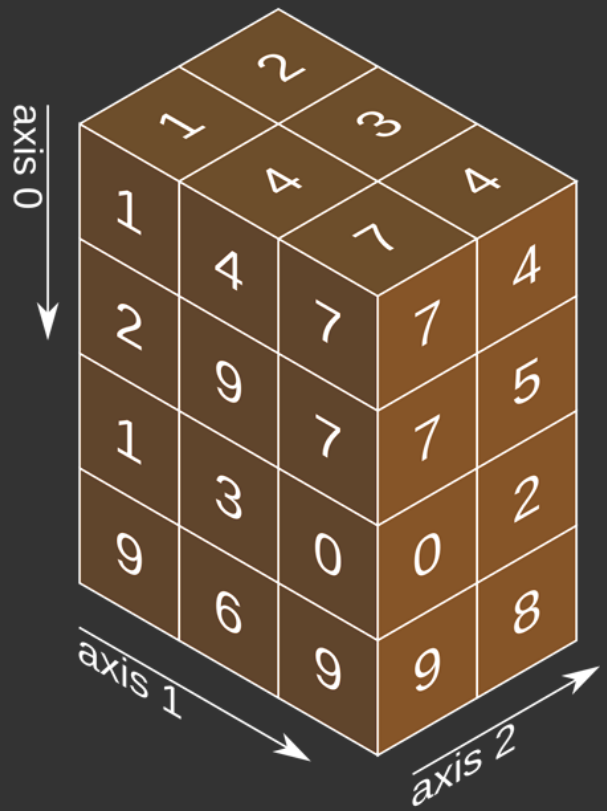
shape: (4,)

2D array



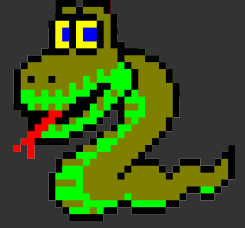
shape: (2, 3)

3D array



shape: (4, 3, 2)

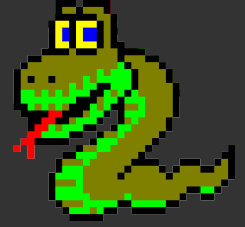
>>> Operations



* Two matrices of the same size can be added.
* Each element of the resulting matrix is the sum of one element of the first marix with the element in the same position in the scond matrix.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$$

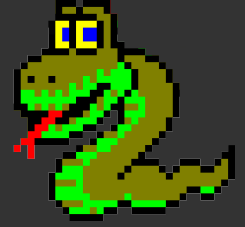
>>> Operations



* Scalar multiplication takes one scalar (a single value) and a matrix.
* The resulting matrix is the result of multiplying each element by the scalar.

$$2 * \begin{pmatrix} 4 & 0 \\ 1 & -9 \end{pmatrix} = \begin{pmatrix} 8 & 0 \\ 2 & -18 \end{pmatrix}$$

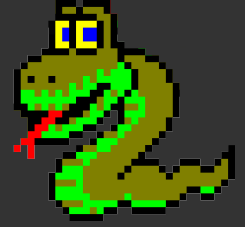
>>> Operations



* Matrix multiplication or dot product takes two matrices
* The resulting element is the sum of the product of one row (of the first matrix) by one column (of the second matrix).

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} * \begin{pmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{pmatrix} = \begin{pmatrix} 58 & \\ & \end{pmatrix}$$

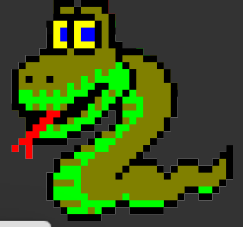
>>> Slicing

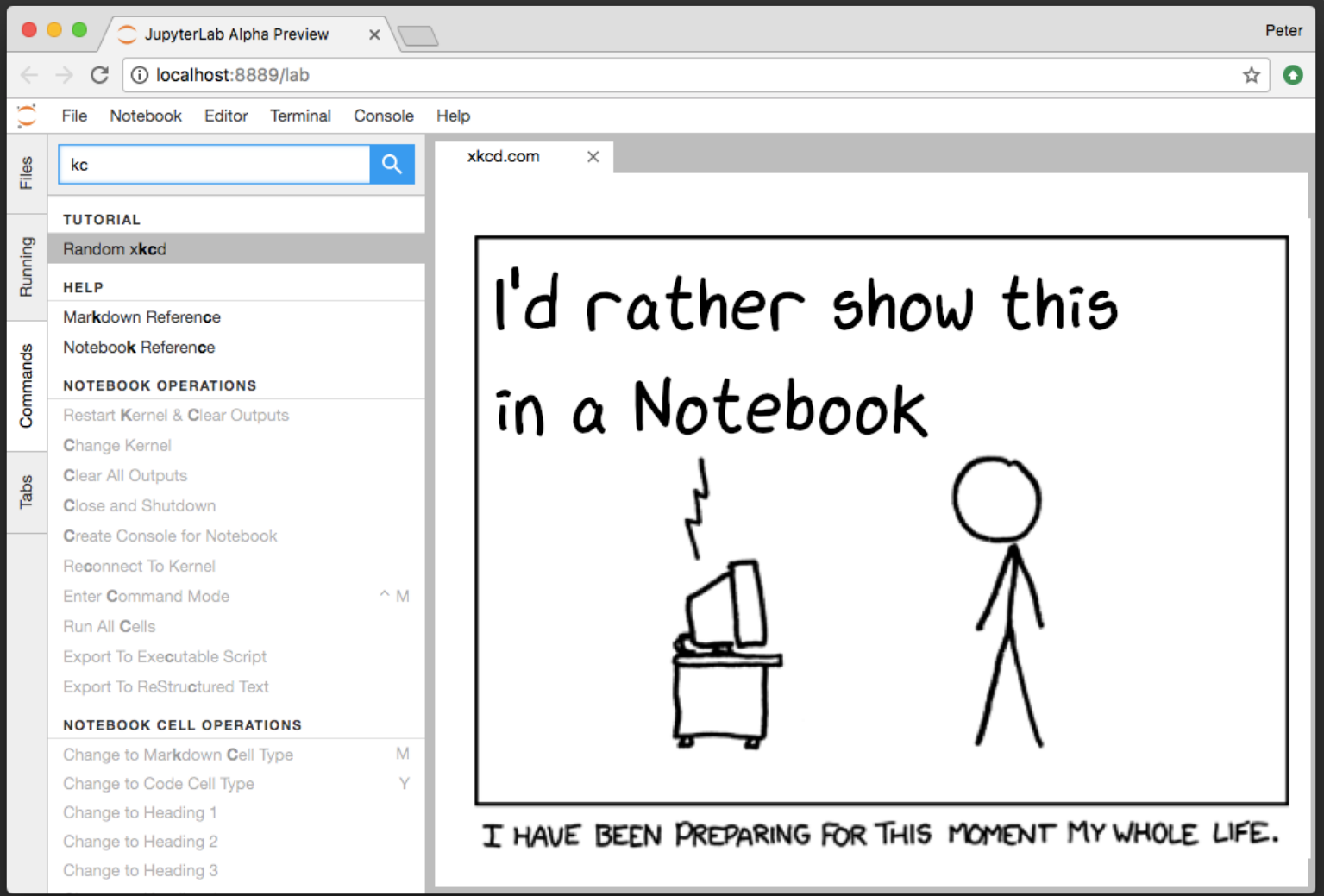


Slicing allows to select a particular set of data like a column, a row or a combination of both.

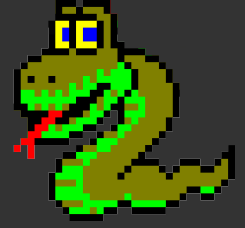
```
>>> a[0,3:5]
array([3,4])


>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

>>> NumPy Demo




JupyterLab Alpha Preview
Peter
localhost:8889/lab
File Notebook Editor Terminal Console Help
Files
Running
Commands
Tabs
kc
TUTORIAL
Random xkcd
HELP
Markdown Reference
Notebook Reference
NOTEBOOK OPERATIONS
Restart Kernel & Clear Outputs
Change Kernel
Clear All Outputs
Close and Shutdown
Create Console for Notebook
Reconnect To Kernel
Enter Command Mode ^ M
Run All Cells
Export To Executable Script
Export To ReStructured Text
NOTEBOOK CELL OPERATIONS
Change to Markdown Cell Type M
Change to Code Cell Type Y
Change to Heading 1
Change to Heading 2
Change to Heading 3
xkcd.com
I'd rather show this in a Notebook
I HAVE BEEN PREPARING FOR THIS MOMENT MY WHOLE LIFE.

>>> Intro to Pandas



Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

* Load data from different sources.
* Clean up and data filtering.
* Extraction, transformation and loading operations.

https://pandas.pydata.org/

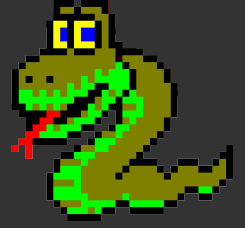
```
$ pip install pandas
```

```
import pandas as pd
```

>>> What is pandas?

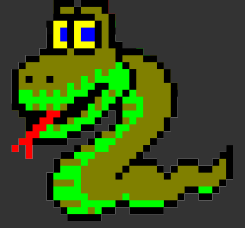
* Pandas goal is to provide fast, flexible, and expressive data structures
* Designed to work with "relational" or "labeled"
* Most common use caseses are:
    * Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
    * Ordered and unordered (not necessarily fixed-frequency) time series data.
    * Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels

>>> What is pandas good for?



* Easy handling of missing values.
* Size mutability: columns can be inserted and deleted.
* Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels.
* Powerful, flexible group by functionality to perform split-apply-combine operations on datasets.
* Works well with foreign data.
* Joining and merging operations.

>>> Dataframe and series

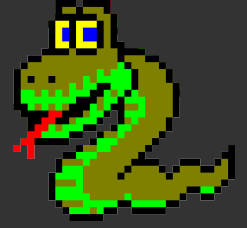


Pandas data frames has two main data structures

* Series: 1D labeled homogeneously-typed array
* DataFrame: General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column

	Birth Month	Origin	Age	Gender
Carly	January	UK	27	f
Rachel	September	Spain	28	f
Nicky	September	Jamaica	28	f
Wendy	November	Italy	22	f
Judith	February	France	19	f

>>> Series

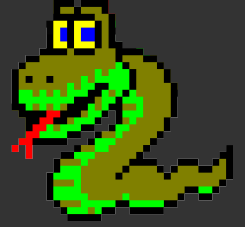


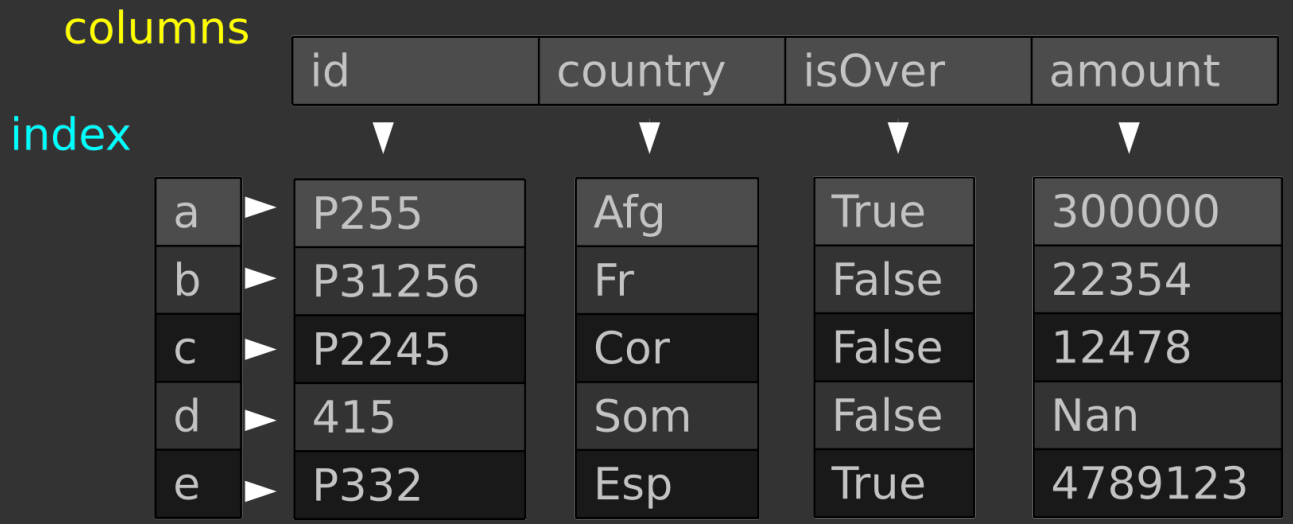
	index	value
0	C	3
1	B	7
2	A	4
3	D	4
4	D	0.3

* Any type of data int, str, float.
* Indexes do not need to be in order
* Indexes do not need to be unique

* Series.index: list of indices
* Series.values: list of values

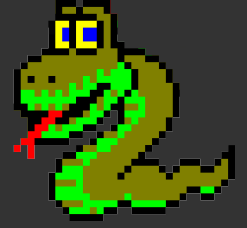
>>> DataFrame





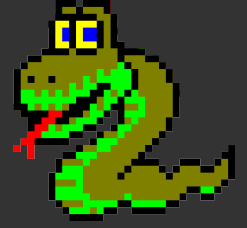
* ndarray like
* 2D data structure (supports nD with multi-index)
* Dictionary of series
* Row and column index
* Size mutable: insert or delete columns

>>> DataFrame



* Some vocabulary:
  * DataFrame.index: list of DataFrame indices
  * DataFrame.values: 2D array of all values contained in the DataFrame
  * DataFrame.columns: list of columns labels
  * axis: indicates the axis index for rows (axis = 0), columns (axis = 1), or even nth axis in multi-index

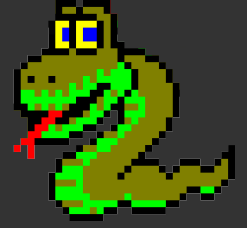
>>> Constructing a DataFrame



* Directly editing

```
s = pd.Series([3,7,4,4,0.3],
              index = ['a','b','c','d','e'])
df = pd.DataFrame(
    np.arange(9).reshape(3,3),
    index = ['b','a','c'], columns=['Paris','Berlin',
    'Madrid']
)
```

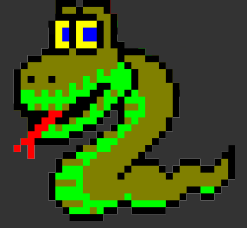
>>> Constructing a DataFrame



* Using dicts

```
data = {
    'Paris': [0,3,6,999999999],
    'Berlin': [1,4,7],
    'Madrid': [2,5,8]
}
df = pd.DataFrame(
    data,
    index = ['b','a','c','d'],
    columns = ['Paris', 'Berlin', 'Madrid']
)
```

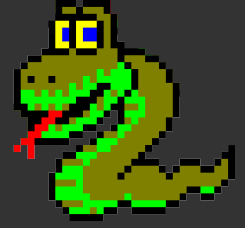
>>> Constructing a DataFrame



* From databases and data files

```
df = pd.read_csv("path/some_file.csv",[index_col = [...]])
df = pd.read_table("path/some_file.txt", [sep = ','])
# SQL
conn = sqlite3.connect(...)
sql = """
  SELECT columns
  FROM tables
  WHERE conditions
"""
df = pd.read_sql_query(sql, conn)
```

>>> Selecting Data



```
In:s
Out:
a    3.0
b    7.0
c    4.0
d    4.0
d    0.3
```

```
In:s['b']
Out:
7.0
```

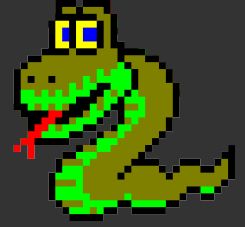
```
In:s['a':'c']
Out:
a    3.0
b    7.0
c    4.0
```

```
In:s['d']
Out:
d    4.0
d    0.3
```

```
In:s[1]
Out:
7.0
```

* The returned object is either a value, or a subset of the initial series s
* Select some data with integer index OR index label
* Warning: Work only if the index type is not numeric

>>> Filtering Data



```
In: df
Out:
Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
c      6       4       8
```

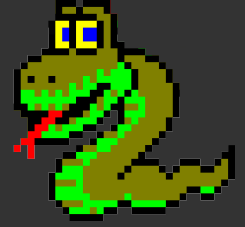
```
In: df[:2]
Out:
Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
```

```
In: df[df['Paris']>1]
Out:
Paris  Berlin  Madrid
a      3       7       5
c      6       4       8
```

```
df.Berlin[df['Berlin']>1]=0
In: df
Out:
Paris  Berlin  Madrid
b      0       1       2
a      3       0       5
c      6       0       8
```

* Returns a subset of the initial DataFrame
* Can be modified
* Conditions can be used for filtering

>>> Filtering Data



* The indexing field ix (deprecated, use instead loc) enables to select a subset of the rows and columns from a DataFrame.

```
In: df
Out:
Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
c      6       4       8
```

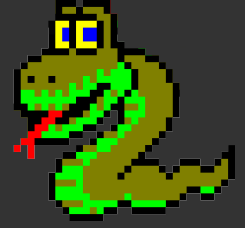
```
In:df.ix['a','Berlin']
Out:
7
```

```
In: df.ix[['b','c'],'Berlin']
Out:
b    1
c    4
Name: Berlin
```

```
In:df.ix[:,'Berlin']
Out:
b    1
a    7
c    4
Name: Berlin
```

* Returns a value OR a Series subset of the DataFrameIn

>>> Dropping data



* On series or DataFrame, drop a row by his index

```
In: s
Out:
a    3.0
b    7.0
c    4.0
d    4.0
d    0.3
```

```
In: s.drop('d')
Out:
a    3.0
b    7.0
c    4.0
```

```
In: s.drop_duplicates()
Out:
a    3.0
b    7.0
c    4.0
d    0.3
```

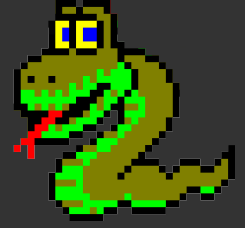
* In DataFrame, (default) 'axis=0' refers to (row) index and axis=1 to columns

```
In: df
Out:
Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
c      6       4       8
```

```
In: df.drop('c')
Out:
Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
```

```
In :df.drop('Berlin', axis=
Out:
Paris  Madrid
b      0       2
a      3       5
c      6       8
```

>>> Data alignment



* Series join and align axis to do operations

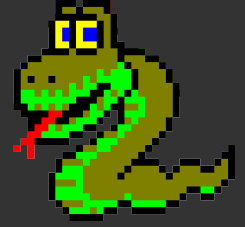
```
In:s
Out:
a    3.0
b    7.0
c    4.0
d    4.0
e    0.3
```

```
In: s2
Out:
a    0

c    1


f    2
```

```
In: s+s2
Out:
a    3
b  NaN
c    5
d  NaN
e  NaN
f  NaN
```

```
In: s.add(s2, fill_value=0)
Out:
a    3.0
b    7.0
c    5.0
d    4.0
e    0.3
f    2.0
```

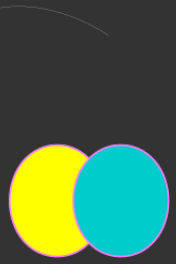
>>> Merge, join, concatenate

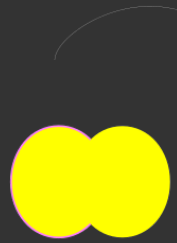


* Many to one

```
In: df1
Out:
   data1 keyLeft
0      0       b
1      1       b
2      2       a
3      3       c
4      4       a
5      5       a
6      6       b
```

```
In: df2
Out:
   data2 key
0      0   a
1      1   b
2      2   d
```





```
In: pd.merge(df1,df2, left_on = 'keyLeft',
right_on='key', how = 'inner')
Out:
   data1 keyLeft  data2  key
0      0       b      1    b
1      1       b      1    b
2      6       b      1    b
3      2       a      0    a
4      4       a      0    a
5      5       a      0    a
```

```
In:pd.merge(df1,df2, left_on =
'keyLeft', right_on='key', how =
'outer')
Out:
   data1 keyLeft  data2  key
0      0       b      1    b
1      1       b      1    b
2      6       b      1    b
3      2       a      0    a
4      4       a      0    a
5      5       a      0    a
6      3       c    NaN  NaN
7    NaN     NaN      2    d
```

>>> Merge, join, concatenate

* Many to many: cartesian product of the rows given a common key

In:left1
Out:

	key	data1
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

In:right1
Out:

	data2	key
0	0	a
1	1	b
2	2	a
3	3	b
4	4	d

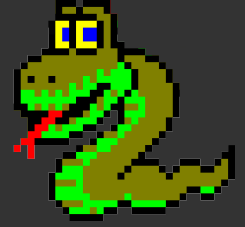
In: pd.merge(left1,right1, on='key', how = 'left')
Out:

	key	data1	data2
0	a	0	0
1	a	0	2
2	a	2	0
3	a	2	2
4	a	3	0
5	a	3	2
6	b	1	1
7	b	1	3
8	b	4	1
9	b	4	3
10	c	5	NaN

In:pd.merge(left1,right1, on='key', how = 'right')
Out:

	key	data1	data2
0	a	0	0
1	a	2	0
2	a	3	0
3	a	0	2
4	a	2	2
5	a	3	2
6	b	1	1
7	b	4	1
8	b	1	3
9	b	4	3
10	d	NaN	4

>>> Series sorting



* Order method: only on Series

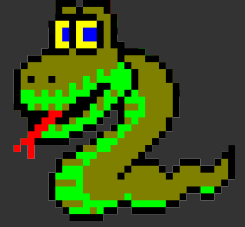
```
In: s
Out:
a    3.0
b    7.0
c    4.0
d    4.0
d    0.3
```

```
In: s.order([ascending=True])
Out:
d    0.3
a    3.0
c    4.0
d    4.0
b    7.0
```

* Sort method by index

```
In:
s.sort_index(ascending=False)
Out:
d    0.3
d    4.0
c    4.0
b    7.0
a    3.0
```

>>> DataFrame sorting



* No order method for DataFrame: specify the axis

```
In: df
Out:
```

	Paris	Berlin	Madrid
b	0	1	2
a	3	7	5
c	6	4	8

```
In:df.sort_index([ascending=True])
Out:
```

	Paris	Berlin	Madrid
a	3	7	5
b	0	1	2
c	6	4	8

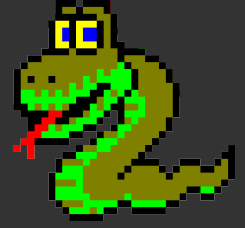
```
In:df.sort_index(by = 'Berlin')
Out:
```

	Paris	Berlin	Madrid
b	0	1	2
c	6	4	8
a	3	7	5

```
In: df.sort_index(axis=1)
Out:
```

	Berlin	Madrid	Paris
a	7	5	3
b	1	2	0
c	4	8	6

>>> Applying a function



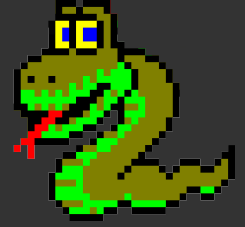
* Applying a function on DataFrame values

```
In: df
Out :
Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
c      6       4       8
```

```
f = lambda x: math.sqrt(x)
In: df.applymap(f)
Out:
Paris     Berlin     Madrid
b  0.000000  1.000000  1.414214
a  1.732051  2.645751  2.236068
c  2.449490  2.000000  2.828427
```

```
df.Berlin = df['Berlin'].map(f)
In: df
Out:
Paris    Berlin    Madrid
b      0   1.000000      2
a      3   2.645751      5
c      6   2.000000      8
```

>>> Statistical values



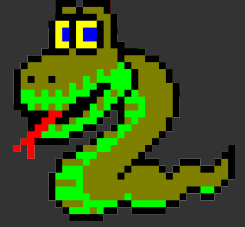
* Objects are equipped with a set of common statistical methods.

```
In: df
Out:
   Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
c      6       4       8
```

```
In: df.sum(axis=1)
Out:
b     3
a    15
c    18
```

```
In: df.describe()
Out:
       Paris  Berlin  Madrid
count    3.0     3.0     3.0
mean     3.0     4.0     5.0
std      3.0     3.0     3.0
min      0.0     1.0     2.0
25%      1.5     2.5     3.5
50%      3.0     4.0     5.0
75%      4.5     5.5     6.5
max      6.0     7.0     8.0
```

>>> Working on indexes



* Reindex Series and DataFrame

In: df
Out:

	Paris	Berlin	Madrid
b	0	1	2
a	3	7	5
c	6	4	8

In:df.reindex(['c','b','a','g'])
Out:

	Paris	Berlin	Madrid
c	6	4	8
b	0	1	2
a	3	7	5
g	NaN	NaN	NaN

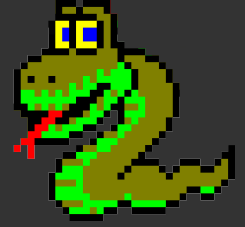
In:df.reindex(['c','b','a','g'], fill_value = 14)
Out:

	Paris	Berlin	Madrid
c	6	4	8
b	0	1	2
a	3	7	5
g	14	14	14

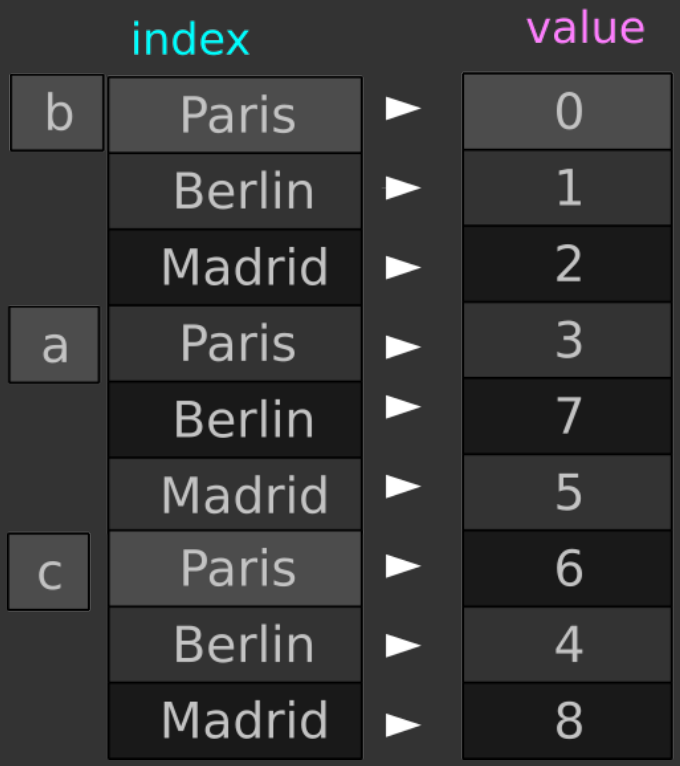
In: df.reindex(columns = ['Varsovie','Paris','Mad
Out:

	Varsovie	Paris	Madrid
b	NaN	0	2
a	NaN	3	5
c	NaN	6	8

>>> Hierarchical indexing

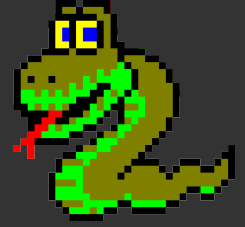


* Indices are n-dimensional tables (n>1)
* Easy to build complex datasets




```
In: df.index
Out: MultiIndex
[(b, Paris), (b, Berlin), (b, Madrid),
(a, Paris), (a, Berlin), (a, Madrid),
(c, Paris), (c, Berlin), (c, Madrid)]
```

>>> Hierarchical indexing



* Build a hierarchical index from DataFrame columns

```
In: df
Out:
Paris   BerlinMadrid
b       0       1 2
a       3       75
c       6       48
```

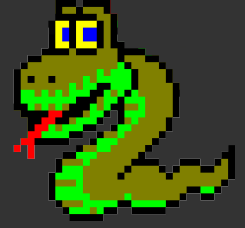
```
df2 = df.set_index(['Berlin','Madrid'])
In: df2
Out:
Paris
BerlinMadrid
12              0
75      ►       3
48      ◄       6       ►
```

* The field xs enables to select values from any index level

```
In: df2.xs(7, level = 0)
Out:
Paris
Madrid
5               3
```

```
In: df2.xs(8, level = 1)
Out:
Paris
Berlin
4               6
```

>>> Hierarchical indexing



* Conversion in Series/DataFrame with methods stack()/unstack()

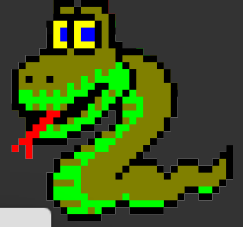
```
In: df
Out:
Paris   Berlin   Madrid
b       0        1        2
a       3        7        5
c       6        4        8
```

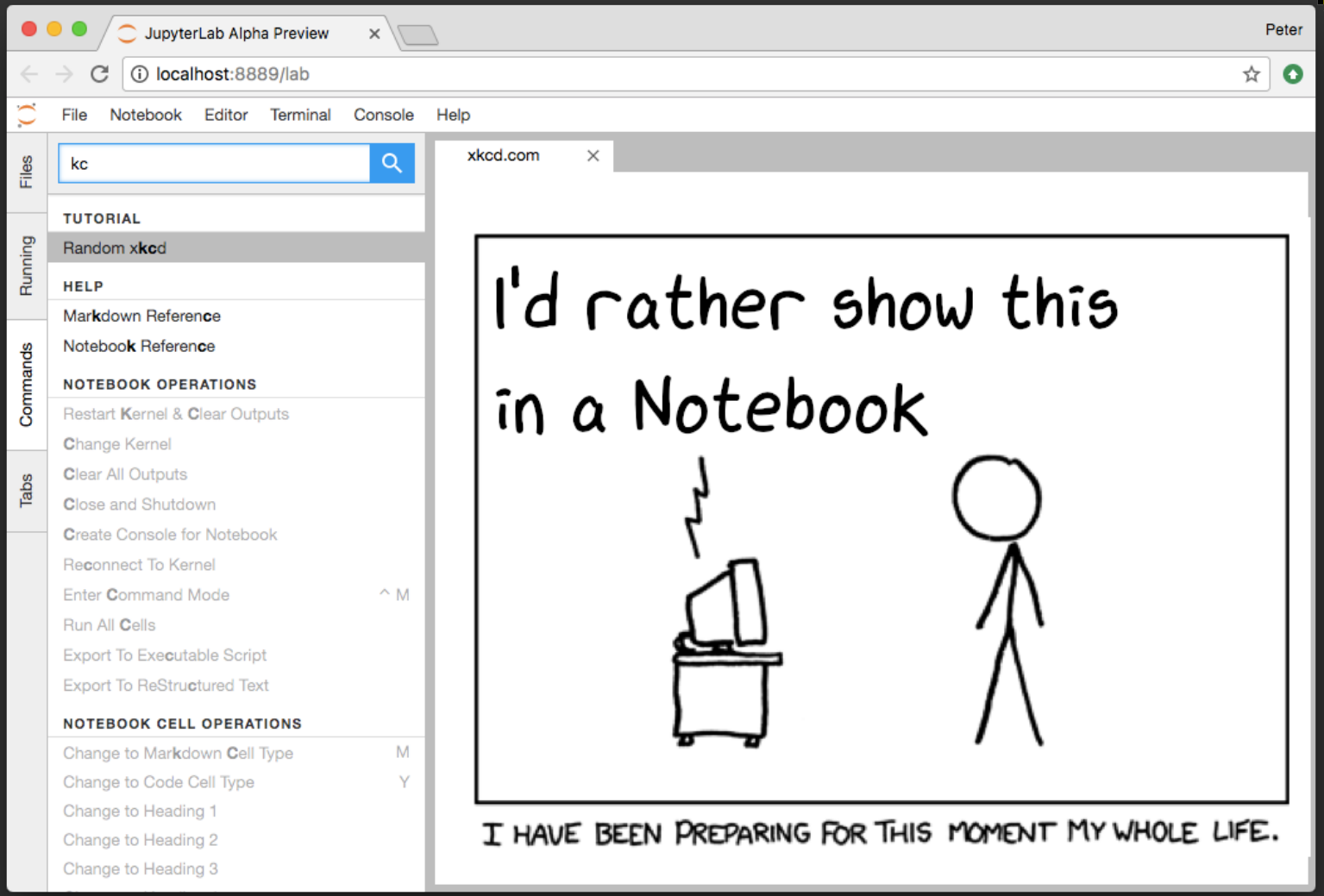
```
In : df.unstack()
Out :
Paris     b     0
a     3
c     6
Berlin    b     1
a     7
c     4
Madrid    b     2
a     5
c     8
```

```
In: df.stack()
Out:
b   Paris      0
Berlin     1
Madrid     2
a   Paris      3
Berlin     7
Madrid     5
c   Paris      6
Berlin     4
Madrid     8
```

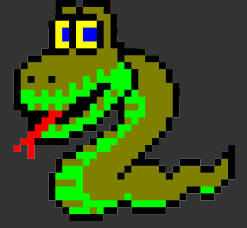
```
In: df.unstack().unstack()
Out:
bac
Paris     036
Berlin    174
Madrid    258
```

>>> Pandas Demo




JupyterLab Alpha Preview
Peter
localhost:8889/lab
File Notebook Editor Terminal Console Help
Files
Running
Commands
Tabs
kc
TUTORIAL
Random xkcd
HELP
Markdown Reference
Notebook Reference
NOTEBOOK OPERATIONS
Restart Kernel & Clear Outputs
Change Kernel
Clear All Outputs
Close and Shutdown
Create Console for Notebook
Reconnect To Kernel
Enter Command Mode ^ M
Run All Cells
Export To Executable Script
Export To ReStructured Text
NOTEBOOK CELL OPERATIONS
Change to Markdown Cell Type M
Change to Code Cell Type Y
Change to Heading 1
Change to Heading 2
Change to Heading 3
xkcd.com
I'd rather show this in a Notebook
I HAVE BEEN PREPARING FOR THIS MOMENT MY WHOLE LIFE.

>>> Things to explore & Gracias!



* Code & slides https://kutt.it/OZf68d
* Scipy Lectures http://scipy-lectures.org/
* Pandas Documentation https://pandas.pydata.org/
* Pandas Examples and Slides
  http://www.renoust.com/pub/presentationPandasPython.pdf