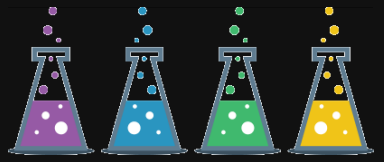
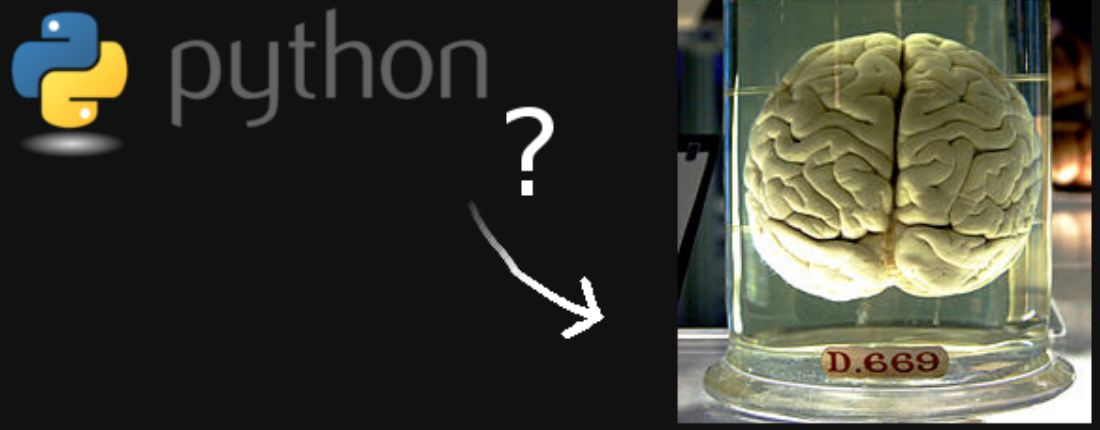
Python in Science

Or how can Python help us understand the human brain?



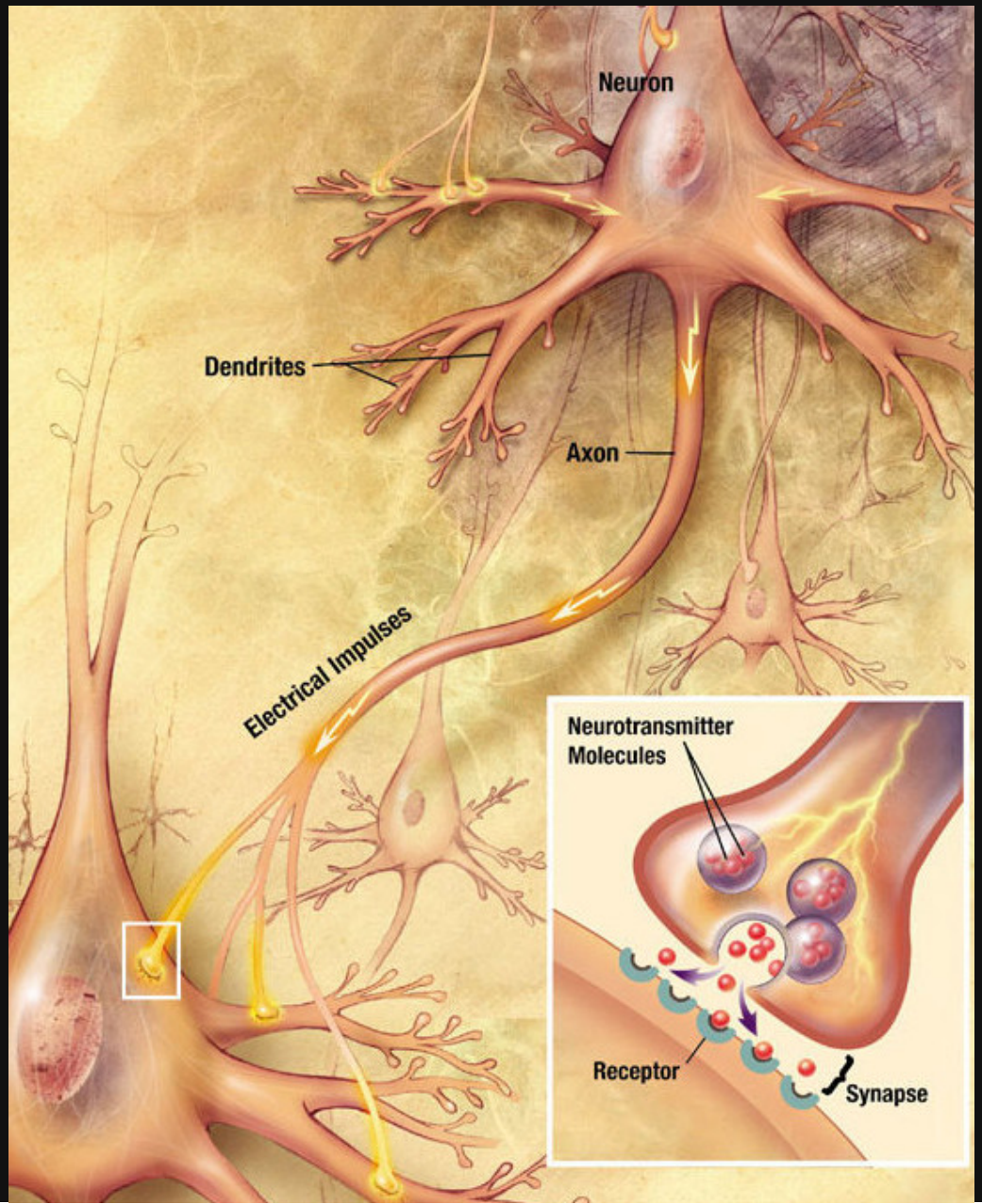
PyLadies Berlin, November 2013

Ivana Kajic Student@TU Berlin | **@kajic_ivana**


python
?
Anthropopithecus troglodytes
D.669

The human brain

- Is only 2% of the weight of the body, but consumes approx. 25% of energy
- Contains 100,000,000,000 neurons which are connected with 60,000,000,000,000 synapses. There are more synapses in the brain than stars in the Milky way!
- Grey brain matter (*neocortex*) is doing "the" job and white brain matter is for communication between neurons
- Unrolling the neocortex would give us around four A4 sheets


Neuron
Dendrites
Axon
Electrical Impulses
Neurotransmitter Molecules
Receptor
Synapse

Can you actually explain how the whole brain works?

- Maybe, but it is very hard. We split the problem of understanding the brain into smaller ones and divide them to various researchers:
  - How does learning work?
  - How is information encoded in the brain?
  - How does human vision work?
  - How do we know that we are happy and not sad?

Neuroscience

- Studies central nervous system (brain + spinal cord)
- Many different subdisciplines that investigate different aspects of human brain
  - Neurophysiology, neuroanatomy, clinical neuroscience, neurolinguistics, neuroeconomics and computational neuroscience…
- Computational neuroscience is a young discipline that uses math, computer science, physics and psychology to explain mechanisms in the human brain

Mind reading

Work in progress in neurosciences: Is it possible to see what someone is thinking?

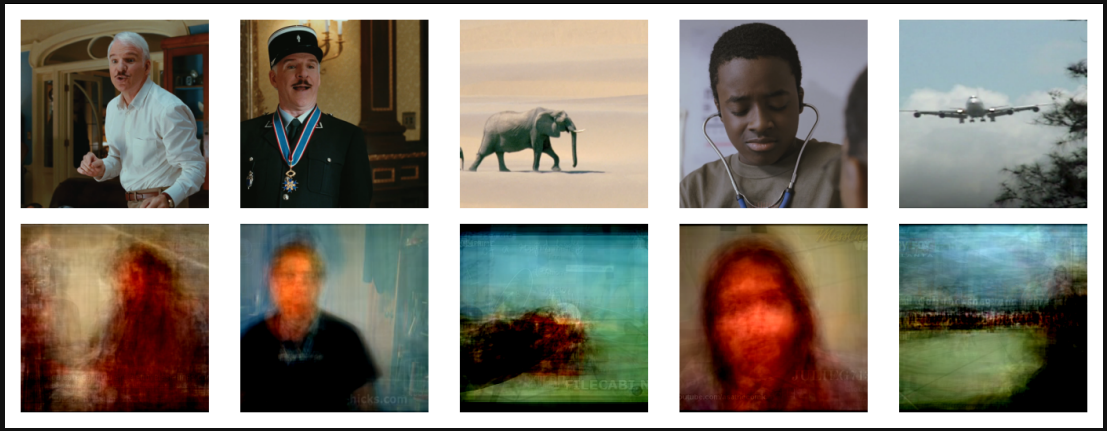
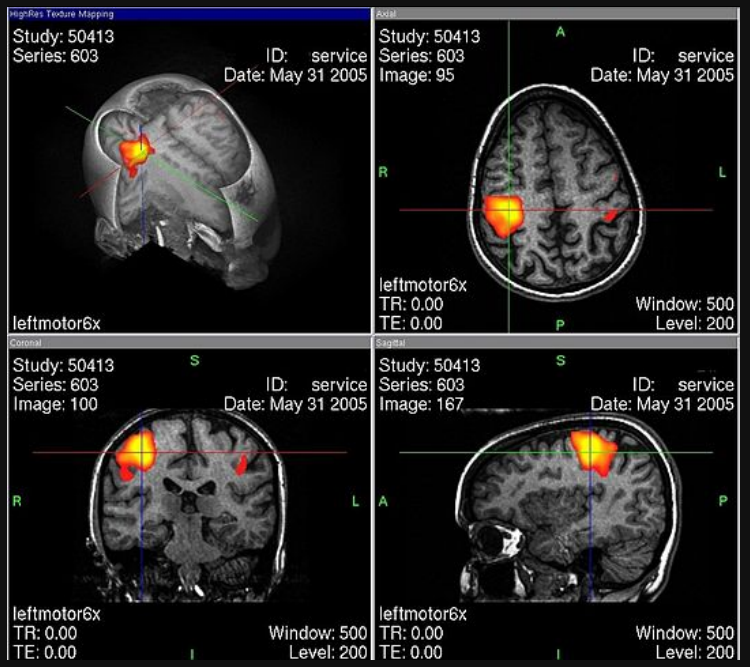


Image and research by **Jack Gallant's Lab** at the UC Berkeley

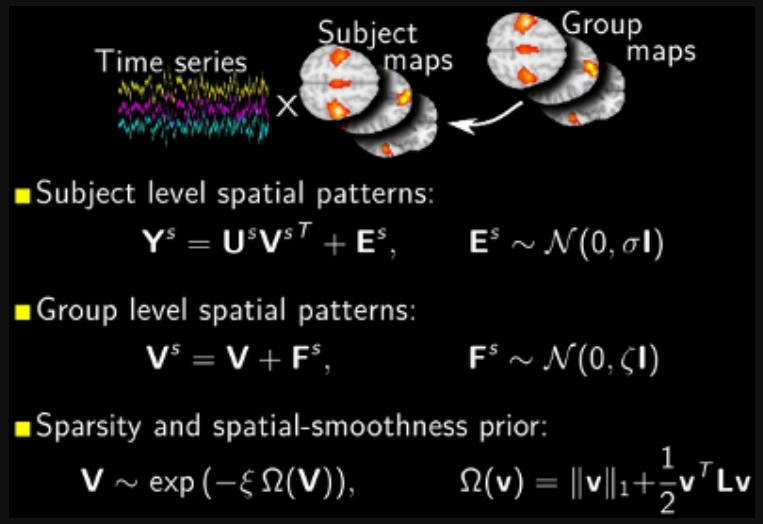
Mind reading: get your data!

- Brain recording using fMRI scanners



Mind reading: make sense out of data

- Make computational models and use them to make predictions
- This is where computational neuroscience kicks in
- Use of machine learning and data mining algorithms, simulations




```
    # Just a fraction of data used for plotting
    data = data[:,:3][::20]
    d = ax.plot(data[:, 0], data[:,1], data[:,2], c='b', marker='*', linestyle='None', ha=0.4, label='data')

    # Plot activated nodes in green
    act_nod = w[act, :]
    a = ax.plot(act_nod[:, 0], act_nod[:, 1], act_nod[:, 2], c='g', marker='o', alpha = 6, label='neurons', markersize=4)

    # Plot inactivated nodes in red
    in_w = w[inact, :]
    i = ax.plot(in_w[:, 0], in_w[:, 1], in_w[:, 2], c='r', marker='o', alpha = 0.6, lab 'inact. neurons', markersize=6, linestyle='None')
    plt.legend(numpoints=1)

def print_strongest_connections(hebb_weights):
    # print the strongest connections
    for i in xrange(param['nr_rows']):
        for j in xrange(param['nr_cols']):
            maxw = -100
            map2X = 0;
            map2Y = 0;
            for k in xrange(param['nr_rows']):
                for t in xrange(param['nr_cols']):
                    if (hebb_weights[i][j][k][t] > maxw):
                        maxw= hebb_weights[i][j][k][t];
                        map2X = k; map2Y = t;
            string = "(" + str(map2X) + ", " + str(map2Y) + ")  "
            sys.stdout.write(string)
        print ''
```

So where did we need Python in the mind reading example?

- Process the data: e.g. remove noise caused by subject movements in the scanner
- Store and organize the data in some sensible way
- Extract patterns from data and build computational models
- Make pretty plots of our results: helps to understand what is happening and useful for communication among scientists

Why did we choose Python?*

- It's free and easy to use: already there on most *nix systems. Just run `python` in a terminal
- Great support: abundance of well maintained packages, especially tools needed in science
- Readable and clear code: hides low level implementation details while following common coding conventions
- Interactive: introspection of your code on the fly

* MATLAB, R and some other programming languages are also used

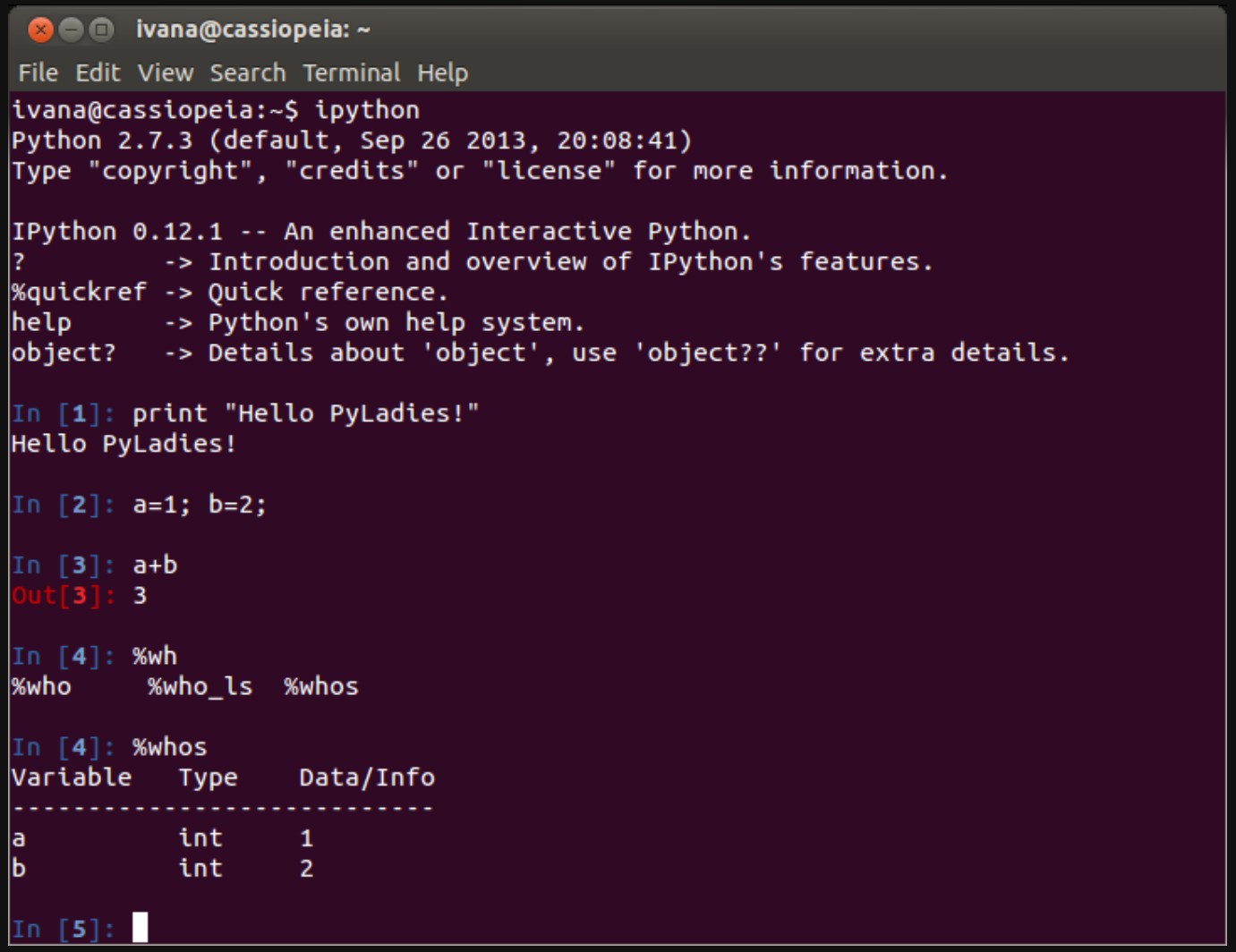
Where Python meets Science

- SciPy Stack is open-source bundle of Python packages for mathematics, science, and engineering
- Some of the included packages in the SciPy stack: NumPy, SciPy library, Matplotlib, IPython, Pandas...
- There are two options to get these packages
  - Download each package separately (instructions)
  - Get the whole set as a single installation file: Anaconda or Canopy

Step 0: IPython

- **IPython** is a simple yet powerful interactive environment to write Python code
- It's similar to the standard Python interpreter, but provides more functionality which makes coding easier and more efficient
- For example: auto-completion using `tab`, save values and restore them later, the history of commands, timing of functions
- IPython **Notebook** is a web-based interactive environment to write and execute code, write text and mathematical formulas, include plots...

IPython


ivana@cassiopeia: ~
File Edit View Search Terminal Help
ivana@cassiopeia:~$ ipython
Python 2.7.3 (default, Sep 26 2013, 20:08:41)
Type "copyright", "credits" or "license" for more information.
IPython 0.12.1 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
In [1]: print "Hello PyLadies!"
Hello PyLadies!
In [2]: a=1; b=2;
In [3]: a+b
Out[3]: 3
In [4]: %wh
%who %who_ls %whos
In [4]: %whos
Variable Type Data/Info
a int 1
b int 2
In [5]:

IPython notebook (web-browser)

IPy spectrogram

https://localhost:8888/8995fc4a-410d-4118-9df8-211274fdce87#

IP[y]: Notebook spectrogram Last saved: Jun 28 10:35 AM Logout

File Edit View Insert Cell Kernel Help

Code

Simple spectral analysis

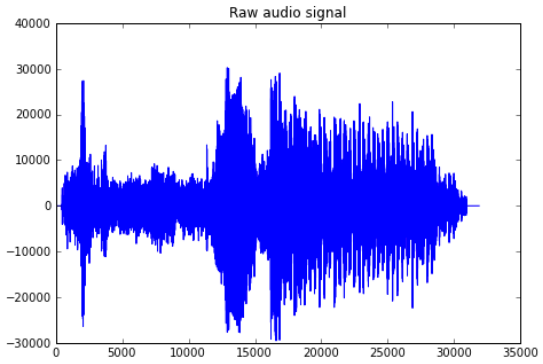
An illustration of the Discrete Fourier Transform

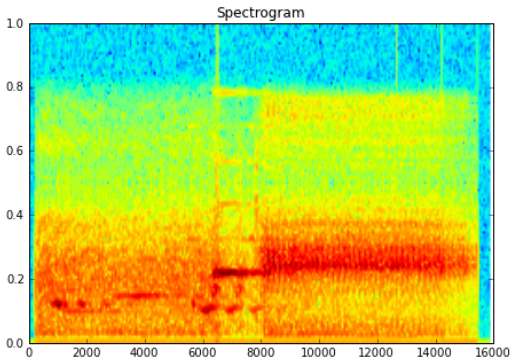
$$X_k = \sum_{n=0}^{N-1} x_n exp^{\frac{-2\pi i}{N} kn} \quad k = 0, \ldots, N-1$$

In [2]:
```
from scipy.io import wavfile
rate, x = wavfile.read('test_mono.wav')
```

And we can easily view it's spectral structure using matplotlib's builtin specgram routine:

In [5]:
```
fig, (ax1, ax2) = plt.subplots(1,2,figsize(16,5))
ax1.plot(x); ax1.set_title('Raw audio signal')
ax2.specgram(x); ax2.set_title('Spectrogram');
```





In [ ]:

Dealing with numbers: NumPy

- **NumPy** is a powerful Python package for efficient manipulation with numbers
- Numbers are "the data": measurements from experiments, results of computer simulations, etc.
- Numbers are stored in containers called `arrays`
- For MATLAB users planning to switch to Python there is a **cheat sheet**

NumPy Array

- Basic data structure used to store numbers
- Vectors are one-dimensional arrays, matrices are two-dimensional, but we can also have N-dimensional arrays
  - Vector containing four numbers: `[1, 2, 3, 4]`
  - Matrix containing six numbers in two rows and three columns:

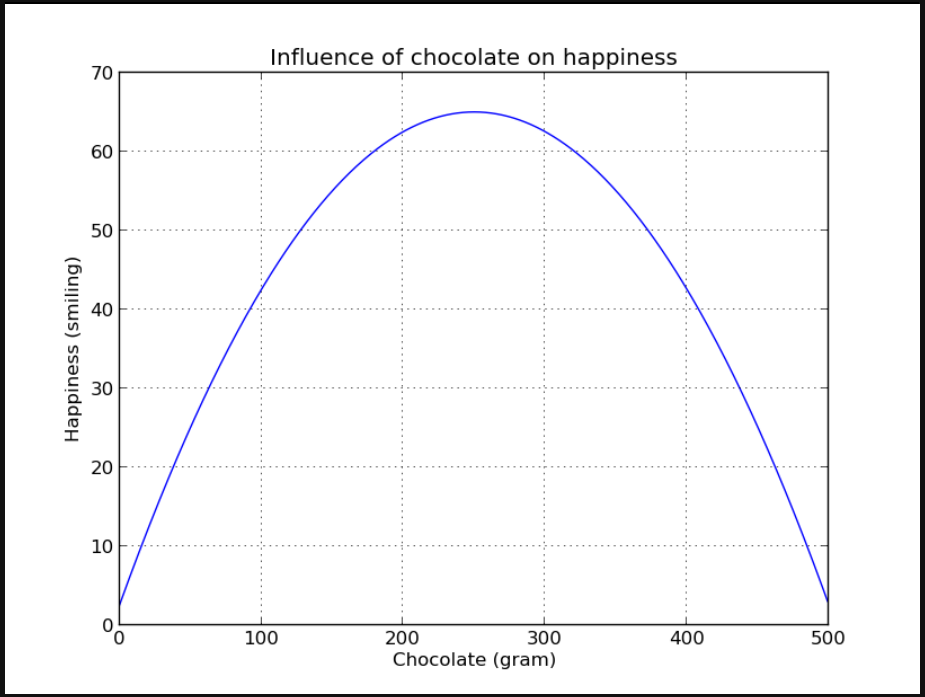
```
[[1, 2, 3],
[4, 5, 6]]
```

- Indexing starts with `0` (not `1`), so the first element in a vector is at the `0`-th position

```
>>> import numpy as np

# Create an array with numbers from 0 to 15
>>> a = np.arange(0, 15)
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])

# What is the first element in the array?
>>> a[0]
0

# What is the shape of our array?
>>> a.shape
(15,)

# Make a new array, this time with rows and columns
>>> b = a.reshape(3,5)
>>> b
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> b.shape
(3, 5)

# Get all the numbers in the second row
>>> b[1, :]
```

- **Matplotlib** is a Python package for data plotting
- Supports different plotting options: histograms, power spectra, bar charts, errorcharts, scatterplots... with just a few lines of code
- Ok, let's plot in Python!

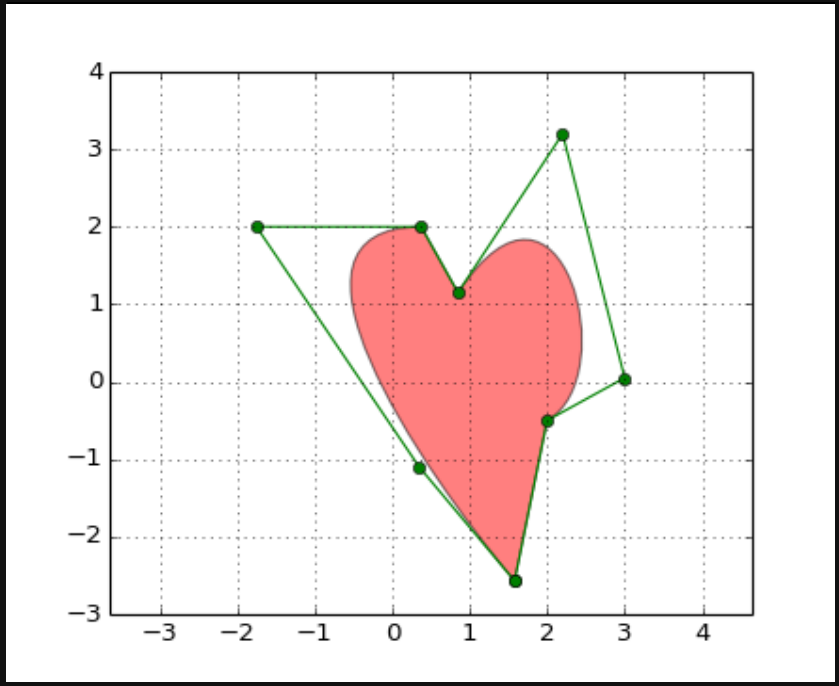
Study: How does the amount of chocolate influence happiness?

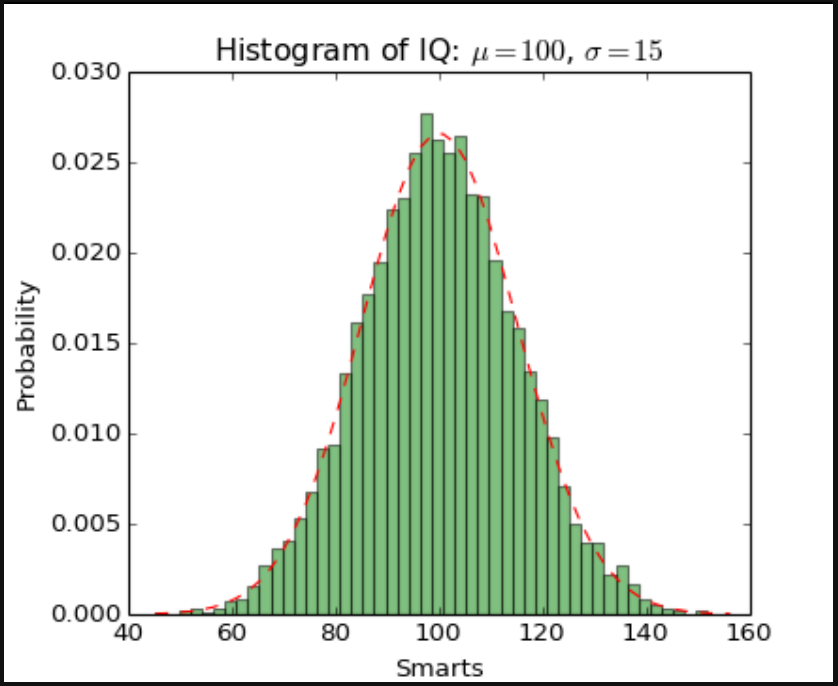


Plotting code

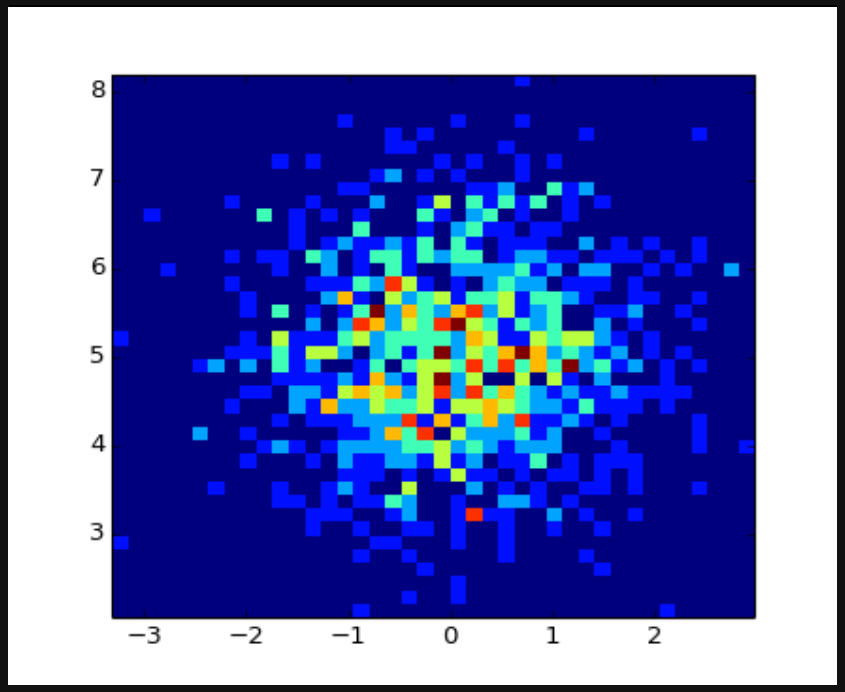
```
>>> import matplotlib.pylab as pl
>>> import numpy as np

# Load data on a drive into two NumPy arrays
>>> chocolate = np.load('chocolate.npy')
>>> happy = np.load('happiness.npy')

# Plot array chocolate vs array happy
>>> pl.plot(choc, happy)

# Label horizontal axis (x) and vertical axis (y)
>>> pl.xlabel('Chocolate (gram)')
>>> pl.ylabel('Happiness (smiling)')

# Give a title to our plot
>>> pl.title('Influence of chocolate on happiness')

>>> pl.grid(True)
>>> pl.savefig("chocolate.png")
>>> pl.show()
```

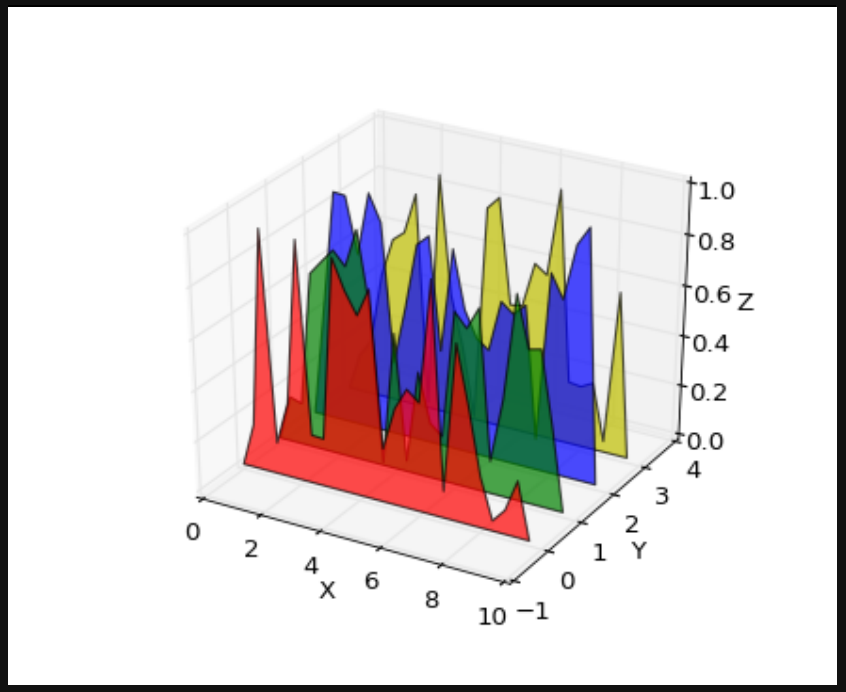
With matplotlib you can do this





... and this!





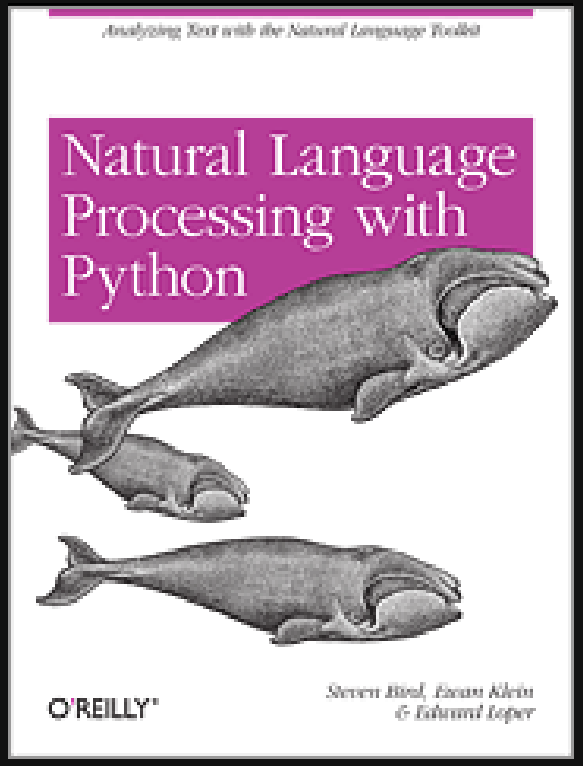
- And much more! These examples are taken from the gallery at the **matplotlib page** which contains many beautiful plots and the source code used to generate them

NLTK

- **Natural language toolkit** suitable for linguists, engineers, students, educators, researchers, and industry users alike
- Suite of text processing libraries for classification, tokenization, stemming, tagging and parsing
- Example applications: RSS Feeds, extracting text from PDF, Word documents, Tweets ...

More on NLTK

- Fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure…
- Available **online** for free:



A peek into the NLTK

```
>>> import nltk
>>> sentence = "Good evening! Would you like some tea?"
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['Good', 'evening', '!', 'Would', 'you', 'like', 'some', 'tea', '?']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged
[('Good', 'NNP'), ('evening', 'NN'), ('!', '.'),
('Would', 'MD'), ('you', 'PRP'), ('like', 'VB'),
('some', 'DT'), ('tea', 'NN'), ('?', '.')]

# NNP and NN is abbreviation for a noun, VB for a verb and PRP for a per
```

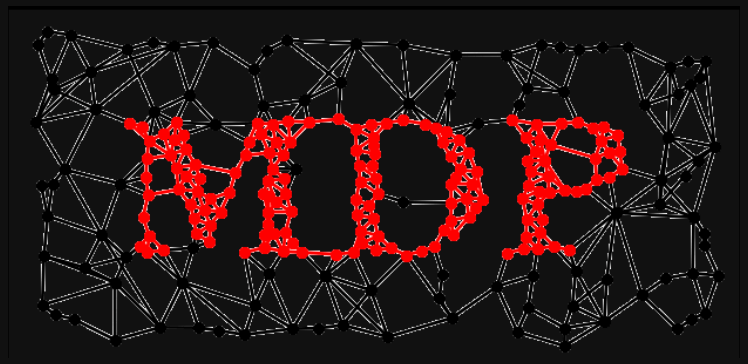
Reading Jane Austen's Emma with NLTK *

- NLTK also includes short excerpts from some books provided by the Project Gutenberg

```
>>> import nltk
>>> nltk.corpus.gutenberg.fileids()
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt',
'bible-kjv.txt', 'blake-poems.txt', 'bryant-stories.txt',
'burgess-busterbrown.txt', ...,
'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt', 'whitman-leaves.txt

>>> emma = nltk.corpus.gutenberg.words('austen-emma.txt')
>>> emma[10:20]
['I', 'Emma', 'Woodhouse', ',', 'handsome', ',', 'clever', ',', 'and', '

>>> print "Number of words:", len(emma)
Number of words: 192427
```

* But I'd rather recommend a cup of tea and a blanket istead of Python for book reading

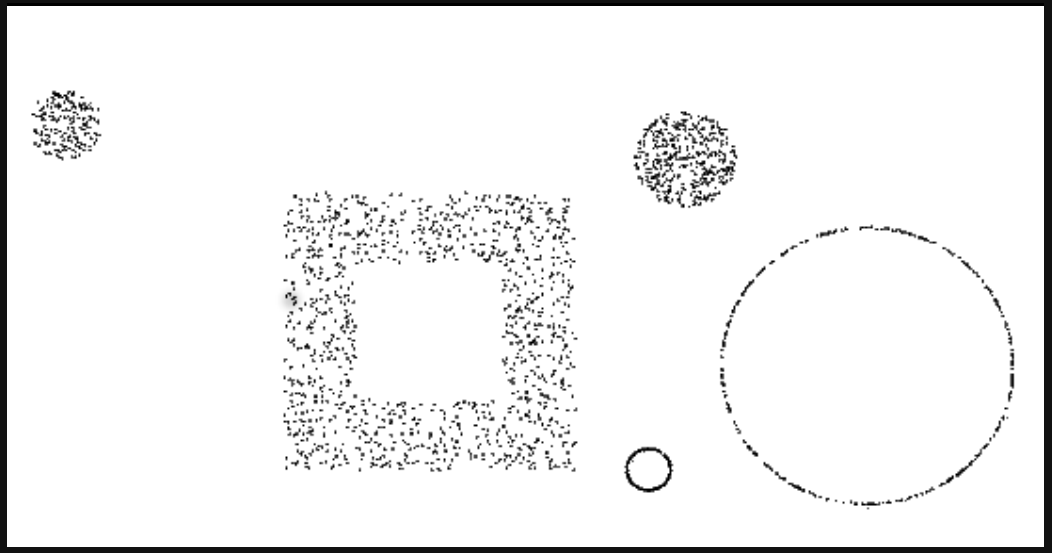
- **Scikit-learn**: satisifes all the data mining and data analysis needs
- Abundance of algorithms for supervised and unsupervised learning
- Classification, regression, clustering, model selection, dimensionality reduction and data preprocessing



- **MDP** is a modular toolkit for data processing
- Set of algorithms for data mining, classification, signal processing, pattern recognition...
- In MDP every algorithm is referred to as a node which can be modified by a user
- Built on NumPy and SciPy

MDP: Detection of objects

- Imagine having 2D geometrical objects such as filled or emtpy circles and rectangles which are made of random points, for example:

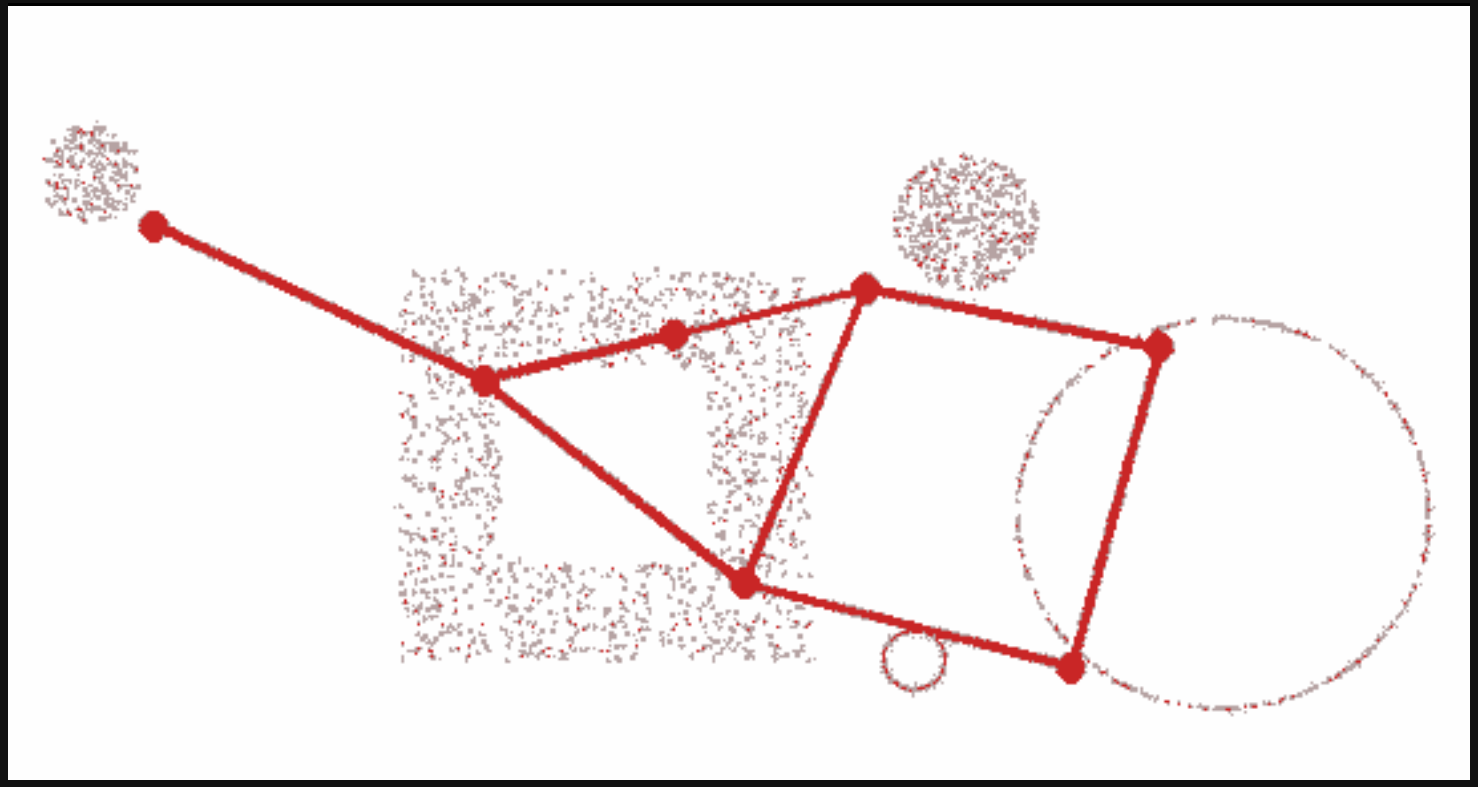


- Our goal is to find out how many objects there are in this image

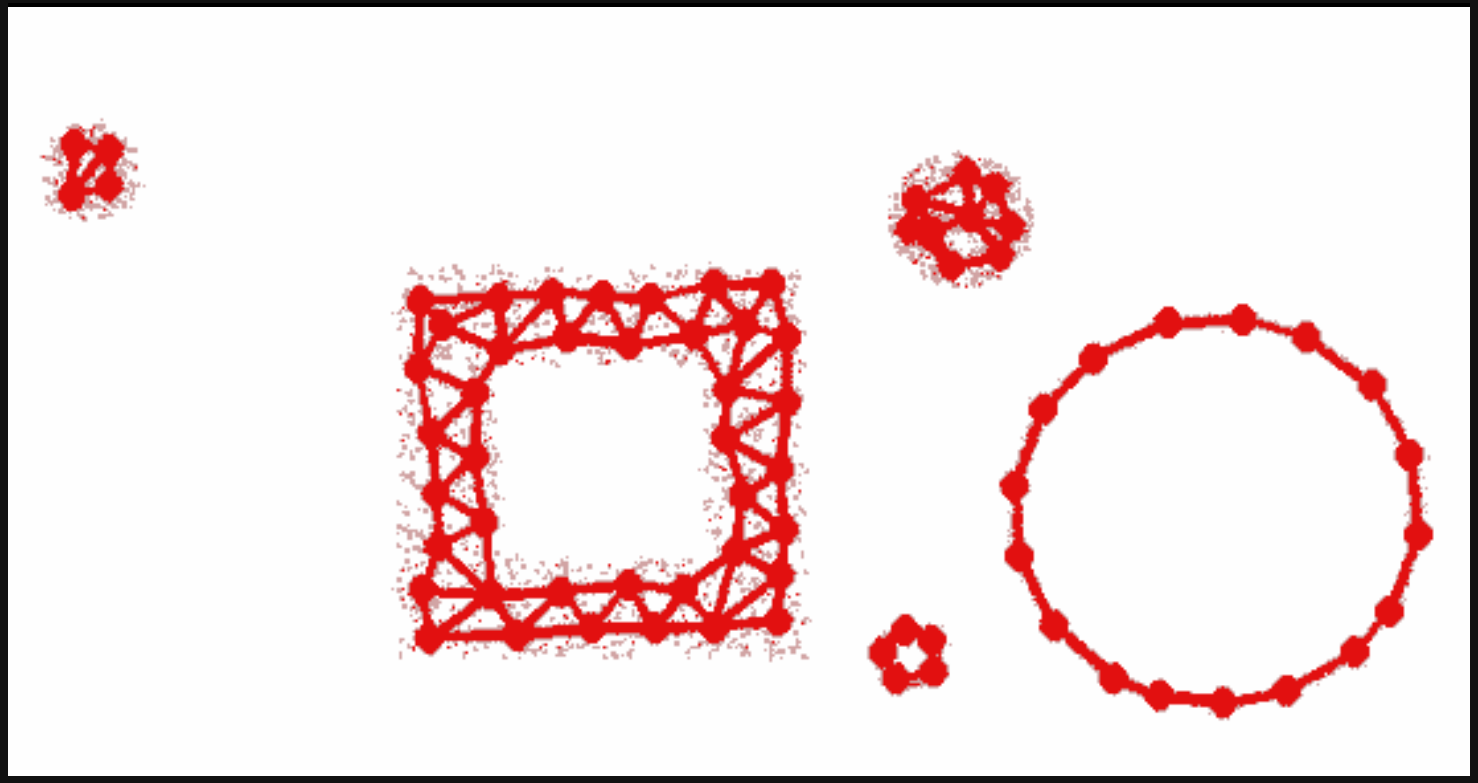
MDP: Detection of objects

- There are many ways to solve this problem, but we love neurons and will use a special type of a neural network called **growing neural gas (GNG)** to solve it
- GNG spreads a sheet of neurons over the data and tries to position neurons so they maximally cover clusters of data
- At every time step, the number of neurons in network increase until some final condition is met

Neural network in the beginning...



... and after the learning!



Code that does what we have just seen:

```
import mdp

x = generate_random_circles_and_rectangles()

gng = mdp.nodes.GrowingNeuralGasNode(max_nodes=75)

STEP = 500
for i in range(0, x.shape[0], STEP):
    gng.train(x[i:i+STEP])
gng.stop_training()

n_obj = len(gng.graph.connected_components())
print n_obj
```

```
Output: 5
```

* Complete source code is available **here**

Python as a glue

- Python can be used to wrap libraries in other programming languages such as C, C++ , Java or Fortran (which are also used in science)
- This is useful when you need to speed up the execution of Python code or the libraries you need don't exist in Python
- **Cython**: Wrap external C libraries
- **f2Py**: Fortran to Python
- **swig**: Turning C code into a Python module

Hands-on session

- There are two tutorials, pick the one you like!
- **Basic tutorial**: Very simple data analysis using NumPy and Matplotlib
  - Recommended for Python beginners
  - The tutorial explains how to install the packages you need
- **Advanced tutorial**: Introduction to machine learning with scikit-learn: Handwritten digit recognition
  - For those of you who already used Python and have basic NumPy experience
  - Installing instructions are available **here**