Giulio Tesei
QDETAILSS WS3
2019-10-24, Lund University

LAYOUT

- What is a Jupyter notebook?
- How can it improve my workflow?
- How to get started
- How to share my notebook to help other scientists to reproduce my analysis

What is it?

Interactive document that integrates:

- code:
  - A long list of available programming languages:
    - Python, Java, R, Julia, Matlab, Octave, Scala, Spark, PHP, C#, C++, etc.

- command-line tools:
  - copying / deleting / moving files with `cp` / `rm` / `mv`
  - navigate in the directory tree with `cd`
  - create a new folder with `mkdir`

- narrative text:
  - equations
  - tables
  - links

- visualizations

Code

Documentation accessible within the notebook.

- How can I call this function?
- Which arguments does it have?
- What attributes does this object have?

In [1]:
```
names = ['marie_curie','amedeo_avogadro','rosalind_franklin']
print(type(names), names[0], type(names[0]))
```

In [2]:
```
for name in names:
    first_last = name.split('_')
    print('first_last is ',first_last)
    first.swapcase() = first_last[0]
    last = first_last[1]
    print(first.capitalize()+' '+last.swapcase())
```

Command-Line Tools:

No need to use the terminal or file managers.

- copy / delete / move files or folders with `cp` / `rm` / `mv`
- navigate in the directory tree with `cd`
- create a new folder with `mkdir`
- check the pth of the current directory with `pwd`

In [ ]:
```
%pwd
```

In [ ]:
```
%mkdir data
%ls
```

In [ ]:
```
%rm -r data
%ls
```

Narrative Text

Markdown markup language:

- equations
- tables
- links

```
###### Free Induction Decay

The oscillating voltage, $V(t)$, has an initial amplitude $V(0)$ which freely decays in time, $t$. Th
e dampened signal can be modeled as a sine function of frequency $\nu$, decaying exponentially with d
ecay constant $T_2$:

\begin{equation}
V(t)=V(0)\,\exp{(-t/T_2)}\,\sin{(2 \pi \nu t)}.
\end{equation}

| Variable      | Description    | Unit  |
|---------------| ---------------|-------|
| $t$       | time | ms |
| $V$       | voltage | V |
| $\nu$     | frequency | Hz |
| $T_2$ | decay constant | ms |

###### References
1. [Wikipedia](http://tiny.cc/r9r0ez)
2. [Merriam-Webster](http://tiny.cc/uas0ez)
```

Free Induction Decay

The oscillating voltage, $V(t)$, has an initial amplitude $V(0)$ which freely decays in time, t. The dampened signal can be modeled as a sine function of frequency ν, decaying exponentially with decay constant T_2:

$$V(t) = V(0)\ \exp\left(-t/T_2\right)\ \sin\left(2\pi\nu t\right).$$

Variable	Description	Unit
t	time	ms
V	voltage	V
ν	frequency	Hz
T_2	decay constant	ms

References

1. Wikipedia
2. Merriam-Webster

How can it improve my workflow?

All the steps of your data analysis and visualization in a single document.

- Interactive data exploration and analysis
- Immediate access to documentation: learn coding, readily use new libraries!
- Facilitates iteration:
  - once the notebook is set up, the analysis can be repeated effortlessly with new variables / data sets
- A large set of freely available tools:
  - Python libraries for linear algebra, fitting data, plotting, handling tabular data, image analysis, bioinformatics, spectroscopy, molecular visualization
- Examples

```
<img src="https://upload.wikimedia.org/wikipedia/commons/1/1a/NumPy_logo.svg" wid
th="200" align="left"/>
```

In [3]:
```
import numpy as np
x = np.linspace(0,np.pi/2.,10)
x
```

Out[3]:
```
array([0.        , 0.17453293, 0.34906585, 0.52359878, 0.6981317 ,
       0.87266463, 1.04719755, 1.22173048, 1.3962634 , 1.57079633])
```

In [4]:
```
np.mean(x) # np.std() to compute the standard deviation
```

Out[4]:
```
0.7853981633974483
```

In [5]:
```
y = np.cos(x) # np.sin(), np.tan(), np.log(), np.exp() etc.
y
```

Out[5]:
```
array([1.00000000e+00, 9.84807753e-01, 9.39692621e-01, 8.66025404e-01,
       7.66044443e-01, 6.42787610e-01, 5.00000000e-01, 3.42020143e-01,
       1.73648178e-01, 6.12323400e-17])
```

In [6]:
```
!head -n 1 aux/pmf.dat
```

```
1.52500000000000000e+01 2.08659275061439544 2e+01 1.22448459294047587 4e-01
```

In [7]:
```
# Load data file
x,y,z = np.loadtxt('aux/pmf.dat',unpack=True)
```

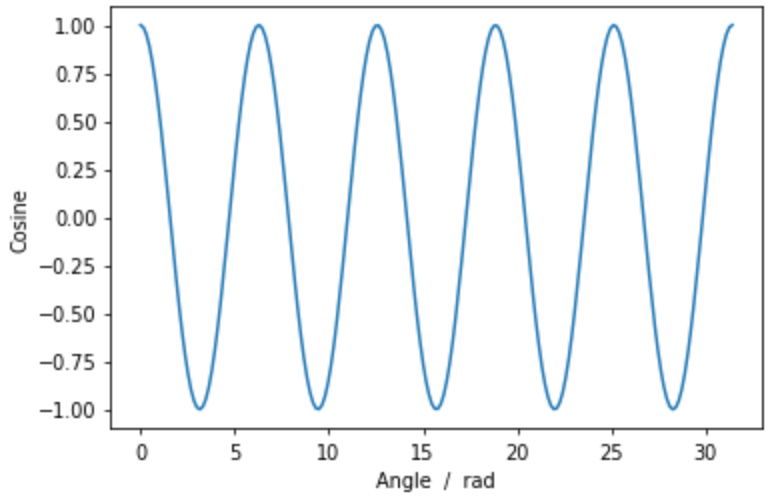
In [8]:
```
# Integrate along the given axis using the composite trapezoidal rule.
np.trapz(y,x)
```

Out[8]:
```
940.9132634027817
```

In [9]:
```
# Return the derivative of an array.
dy = np.gradient(y,x)
# Save the gradient to a text file
np.savetxt('aux/gradient.dat',y)
```

matplotlib

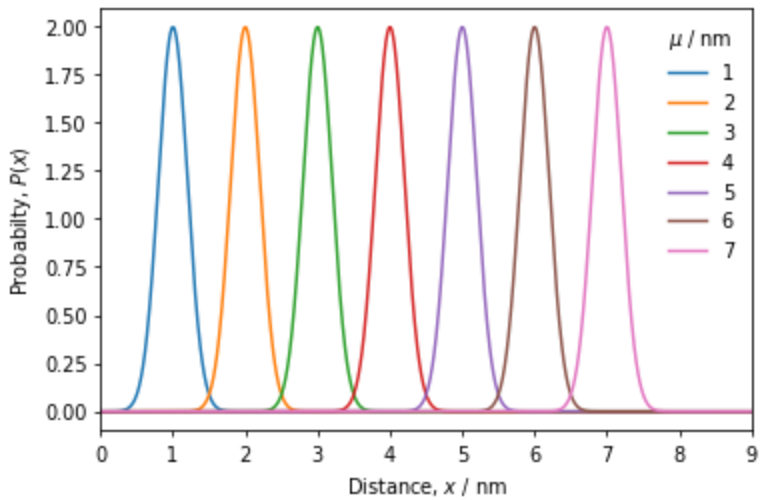
In [57]:
```
import matplotlib.pyplot as plt
x = np.linspace(0,10*np.pi,200)
y = np.cos(x)
plt.plot(x,y)
plt.ylabel('Cosine')
plt.xlabel('Angle  /  rad')
plt.show()
```



$$P(x) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$

In [58]:

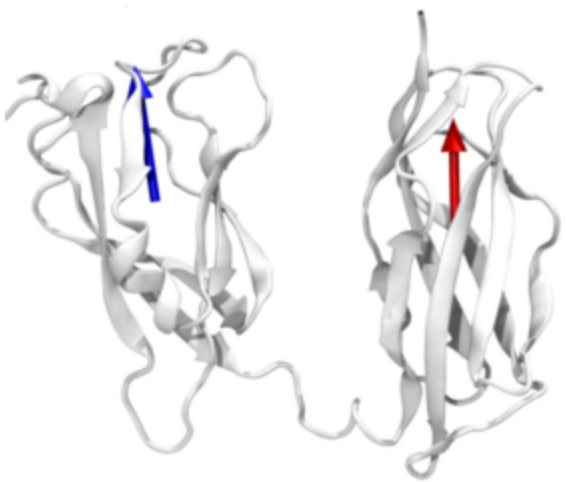
```
x = np.linspace(0,9,1000)
for u in range(1,8):
    y = np.exp(-(u-x)**2/(2*0.2**2)) / np.sqrt(2*np.pi*0.2**2)
    plt.plot(x,y,label=str(u))
plt.legend(frameon=False, title='$\mu$ / nm')
plt.xlim(0,9)
plt.xlabel('Distance, $x$ / nm'); plt.ylabel('Probabilty, $P(x)$')
plt.savefig('aux/normal.pdf') # png, jpg, eps
plt.show()
```



In [53]:

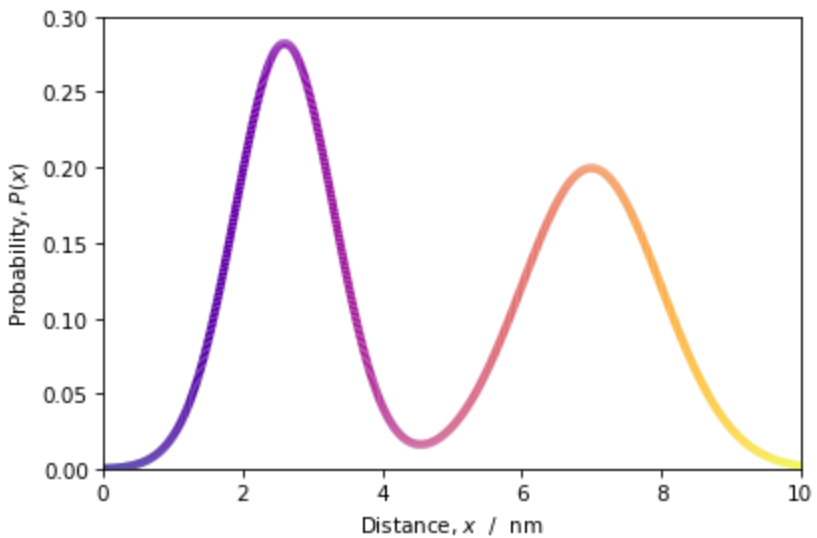
```
import matplotlib.image as mpimg
img = mpimg.imread('aux/protein.png')
print(img.shape)
fig = plt.figure(figsize=(1.2, 1.2))
plt.imshow(img, interpolation='bilinear')
plt.axis('off')
plt.show()
```

```
(900, 1000, 3)
```



In [16]:

```
from matplotlib.collections import LineCollection
x = np.linspace(0,10,1000)
y = np.exp(-(2.6-x)**2) / np.sqrt(4*np.pi) + np.exp(-(7-x)**2/2) / np.sqrt(8*np.pi)
points = np.array([x, y]).T.reshape(-1, 1, 2)
segments = np.concatenate([points[:-1], points[1:]], axis=1)
norm = plt.Normalize(x.min(), x.max())
lc = LineCollection(segments, cmap='plasma', norm=norm)
lc.set_array(x); lc.set_linewidth(4); plt.gca().add_collection(lc)
plt.ylabel(r'Probability, $P(x)$'); plt.xlabel(r'Distance, $x$  /  nm')
plt.xlim(0,10); plt.ylim(0,.3)
plt.show()
```



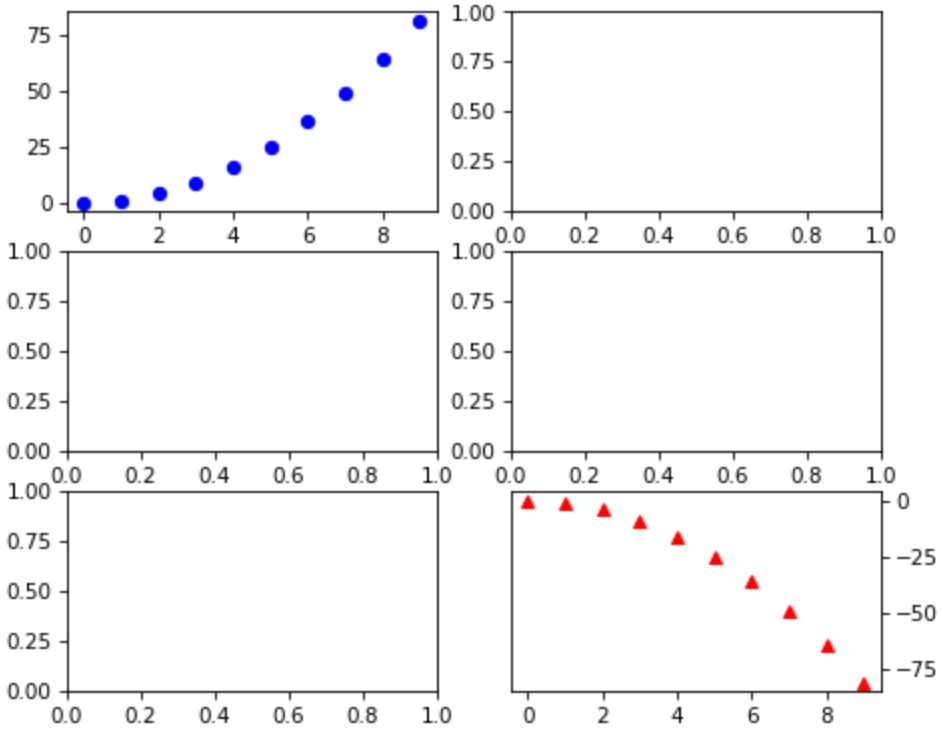
Q: How can I plot a gradient-colored line?

A: Google "matplotlib gradient color line"

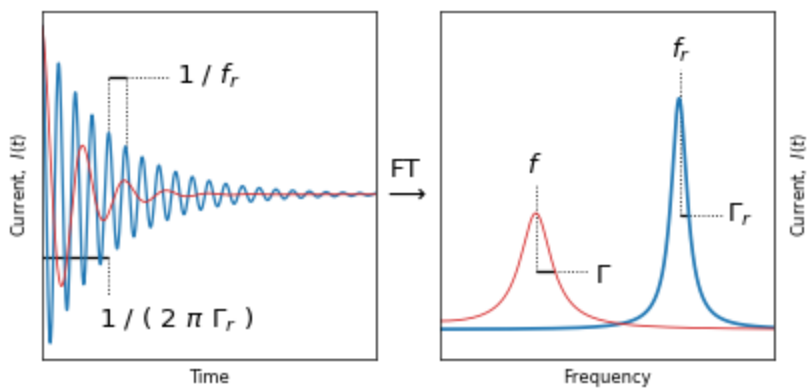
You can google very specific questions and quickly find excellent answers, generally on matplolib.org or stackoverflow.com

Multiple Subplots

In [17]:
```
fig, axes = plt.subplots(nrows=3,ncols=2,figsize=(7, 6))
x = np.arange(10)
axes[0,0].plot(x, x**2, 'bo')
axes[2,1].plot(x, -x**2, 'r^')
axes[2,1].yaxis.set_ticks_position('right') # yticks on the right side
```



In [60]:
```
plotQCM()
plotFourier()
```



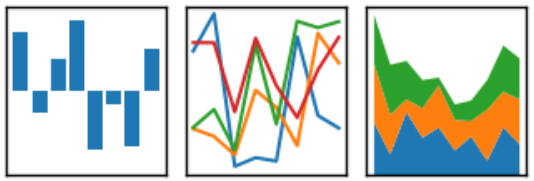
Jupyter Widgets

Gain control and visualize changes in the data!

In [22]:
```
interactive_plot = interactive(plot_cos_decay_FT, freq=(1, 5, .1), gamma=(.08,.14,.01) )
interactive_plot.children[0].description=r'$f$' # slide bar
interactive_plot.children[1].description=r'$\Gamma$' # slide bar
interactive_plot
```

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

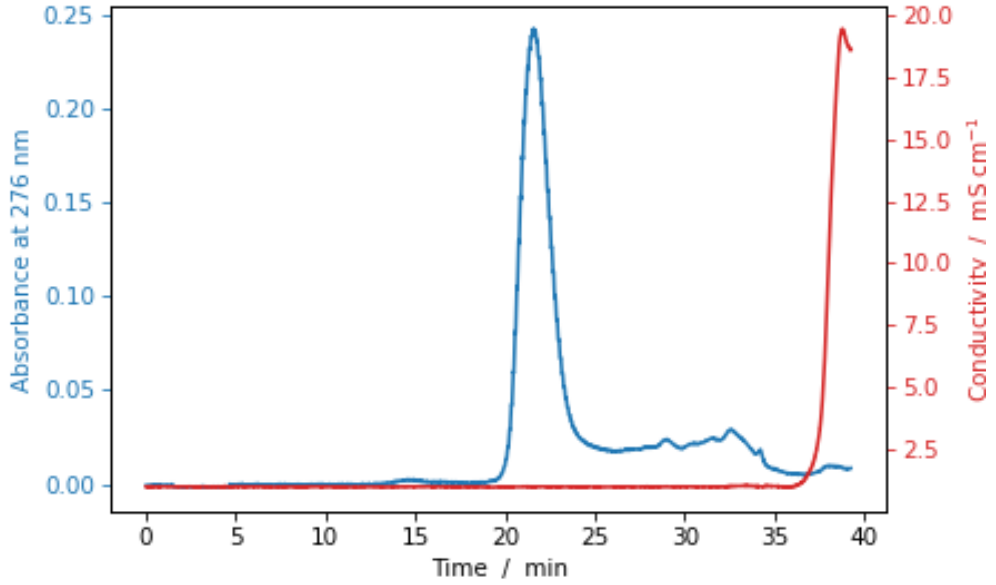


In [23]:
```
import pandas as pd
from IPython.display import display
```

Library to handle tabular data: a convenient alternative to Excel!

Size-Exclusion Chromatography

Data from the purification of α-synuclein monomers kindly provided by **Veronica Lattanzi**



In [24]:

```
%%bash
head -n 22 aux/191923_d_alphasyn_NIST.TXT
```

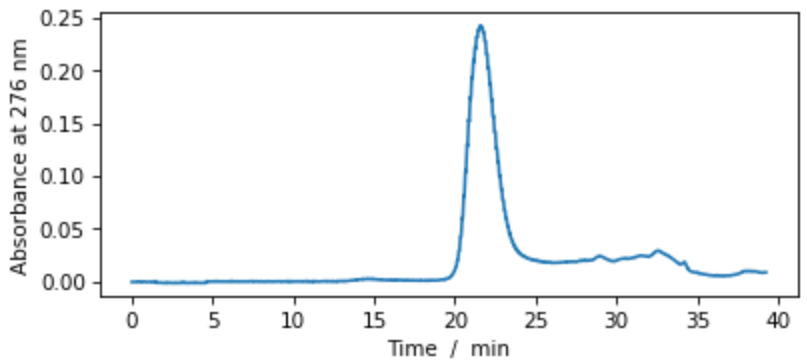
```
Run Name,20191023 dasyn NIST
Run Date,12:10:53 PM   10-23-19
Method Name,Increase_pumpA
Export Format Version, 1.00
Method ID, 2059
Points/Second, 5.00
Number of Records,  11780
Offset from Run Start Time,00:00:00
Run End Time,00:39:17
Time,Second
UV,AU,
Conductivity,mS/cm
Gradient Pump,<Not Selected>
Trace 3,<Not Available>
Trace 4,<Not Available>
Trace 5,<Not Available>
Trace 6,<Not Available>
GP Pressure,<Not Selected>
Volume,ml
Fraction,<Not Available>
Time,UV,Conductivity,Volume
     0.0,-0.000730,   1.040,     0.0
```

In [25]:
```
df = pd.read_csv('aux/191923_d_alphasyn_NIST.TXT',header=20,sep=',',index_col=0)
display(df.head(2))
```

	UV	Conductivity	Volume
Time			
0.0	-0.000730	1.04	0.0
0.2	-0.000724	1.04	0.0

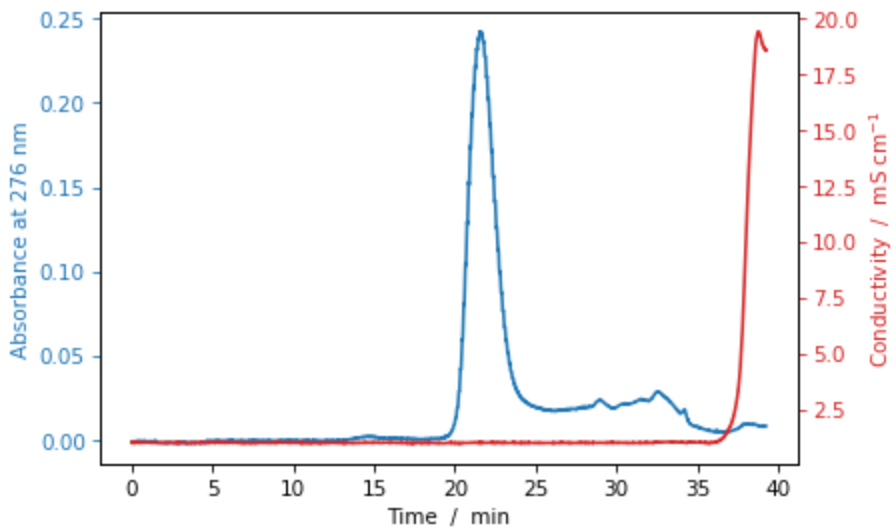
In [26]:
```
fig = plt.figure(figsize=(6, 2.5))
plt.plot(df.index/60, df['UV'])
plt.ylabel('Absorbance at 276 nm'); plt.xlabel('Time  /  min')
```

Out[26]:
```
Text(0.5,0,'Time  /  min')
```



In [27]:

```
fig = plt.figure(); ax1 = plt.axes()
ax1.plot(df.index/60, df['UV'])
ax2 = ax1.twinx() # creates a new subplot identical to x1, with invisible x-axis and y-axis on the r.
h.s
ax2.plot(df.index/60, df['Conductivity'],color=plt.cm.tab10(3))
ax1.tick_params(axis='y',colors=plt.cm.tab10(0))
ax1.set_xlabel('Time  /  min')
ax1.set_ylabel('Absorbance at 276 nm',color=plt.cm.tab10(0))
ax2.set_ylabel('Conductivity  /  mS cm$^{-1}$',color=plt.cm.tab10(3))
ax2.tick_params(axis='y',colors=plt.cm.tab10(3))
plt.savefig('aux/chromatogram2.png')
```



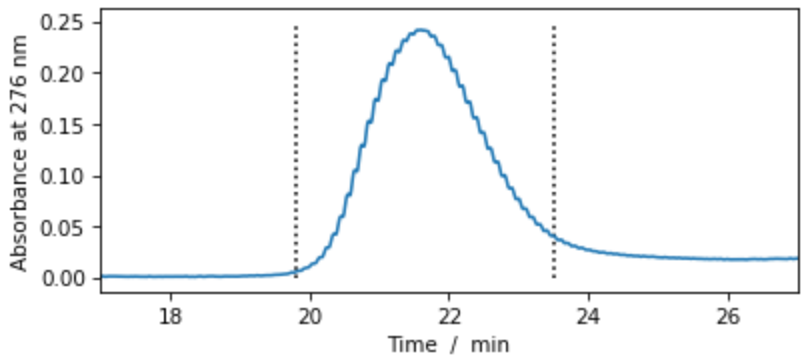
In [28]:

```
fig = plt.figure(figsize=(6, 2.5))

plt.plot(df.index/60, df['UV'])
plt.xlim(17,27)
t1 = 19.8; t2 = 23.5
plt.vlines([t1,t2],ymin=0,ymax=.25,linestyle=':')

plt.ylabel('Absorbance at 276 nm'); plt.xlabel('Time  /  min'); plt.show()

t1 = 19.8*60; t2 = 23.5*60 # convertion to seconds
abs_avg = np.mean(df.loc[t1:t2]['UV']); epsilon = 5960 ;path_length = 0.5
print('Monomer concentration:','{:.3f} µM'.format(abs_avg/epsilon/path_length*1e6))
```




```
Monomer concentration: 42.796 µM
```

Data Scraping: Importing an HTML Table from Sigma Aldrich

In [73]:
```
url = "https://www.sigmaaldrich.com/life-science/metabolomics/learning-center/amino-acid-reference-chart.html"
df = pd.read_html(url, header=0, index_col=0, na_values='-')[1]
df = df['Alanine':'Valine'] # select rows we are interested in
df = df.apply(pd.to_numeric,errors='ignore') # convert numbers from strings to numeric values
display( df.iloc[::3] ) # show every third amino acid
```

Name	3-Letter Symbol	1-Letter Symbol	Molecular Weight	Molecular Formula	Residue Formula	Residue Weight (-H2O)	pKa1	pKb2
Alanine	Ala	A	89.10	C3H7NO2	C3H5NO	71.08	2.34	9.69
Aspartic acid	Asp	D	133.11	C4H7NO4	C4H5NO3	115.09	1.88	9.60
Glutamine	Gln	Q	146.15	C5H10N2O3	C5H8N2O2	128.13	2.17	9.13
Hydroxyproline	Hyp	O	131.13	C5H9NO3	C5H7NO2	113.11	1.82	9.65
Lysine	Lys	K	146.19	C6H14N2O2	C6H12N2O	128.18	2.18	8.95
Proline	Pro	P	115.13	C5H9NO2	C5H7NO	97.12	1.99	10.60
Threonine	Thr	T	119.12	C4H9NO3	C4H7NO2	101.11	2.09	9.10
Valine	Val	V	117.15	C5H11NO2	C5H9NO	99.13	2.32	9.62

In [74]:
```
display( df['Arginine':'Glutamic acid'][['pKa1','pKb2','pKx3']] )
```

	pKa1	pKb2	pKx3
Name			
Arginine	2.17	9.04	12.48
Asparagine	2.02	8.80	NaN
Aspartic acid	1.88	9.60	3.65
Cysteine	1.96	10.28	8.18
Glutamic acid	2.19	9.67	4.25

In [31]:
```
df['Arginine':'Glutamine']['Molecular  Weight'].values
```

Out[31]:
```
array([174.2 , 132.12, 133.11, 121.16, 147.13, 146.15])
```

In [32]:
```
df['Arginine':'Glutamine']['Molecular  Weight'].values.mean()
```

Out[32]:
```
142.31166666666667
```

In [33]:
```
display(df[df['pl4']>7])
```

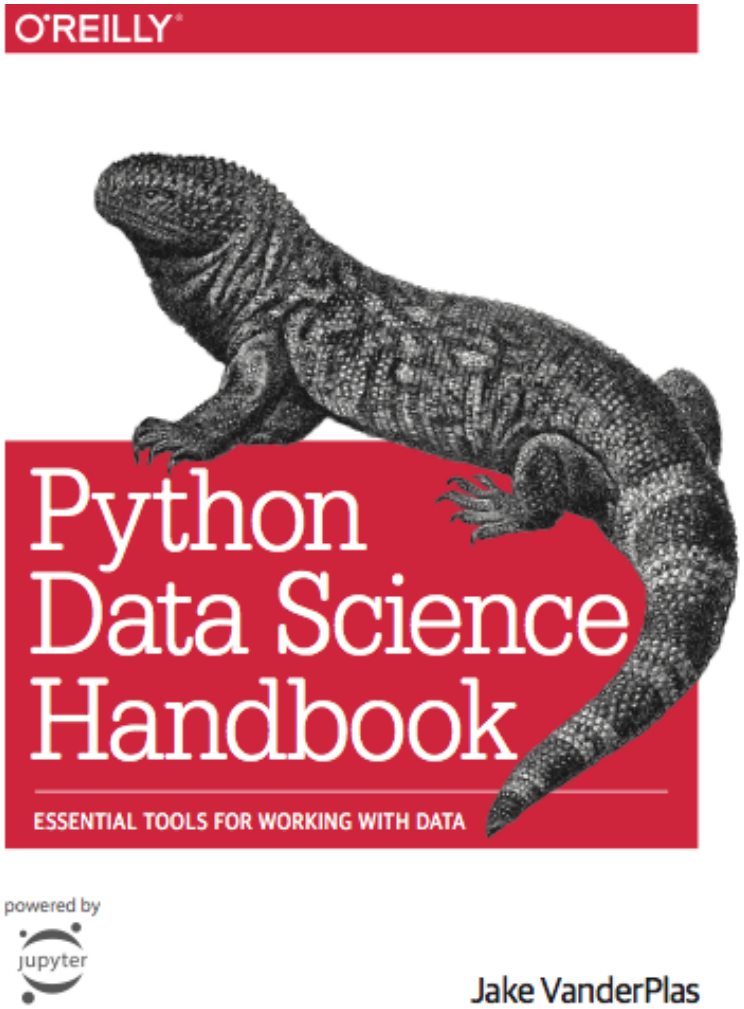
	3-Letter Symbol	1-Letter Symbol	Molecular Weight	Molecular Formula	Residue Formula	Residue Weight (-H2O)	pKa1	pKb2	pKx3
Name									
Arginine	Arg	R	174.20	C6H14N4O2	C6H12N4O	156.19	2.17	9.04	12.48
Histidine	His	H	155.16	C6H9N3O2	C6H7N3O	137.14	1.82	9.17	6.00
Lysine	Lys	K	146.19	C6H14N2O2	C6H12N2O	128.18	2.18	8.95	10.53

In [34]:
```
df[df['pl4']>7]['Molecular  Weight'].values.mean()
```

Out[34]:
```
158.51666666666668
```

In [35]:
```
np.mean(df['Molecular  Weight']-df['Residue Weight  (-H2O)'])
```

Out[35]:
```
18.014545454545452
```



https://jakevdp.github.io/PythonDataScienceHandbook/

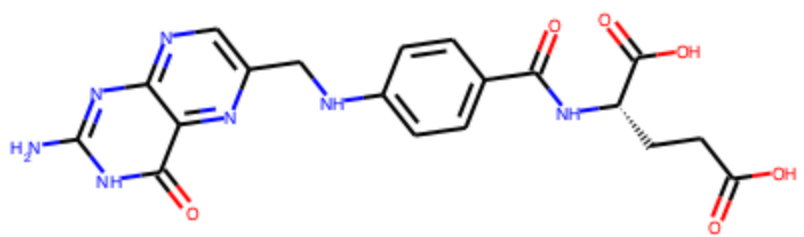
Jupyter Course in Lund

Reproducible and Interactive Data Analysis and Modelling using Jupyter Notebooks (4 ECTS)

- course developed by me, Caterina Doglioni, Mikael Lund and Benjamin Ragan-Kelley
- COMPUTE research school (Natural Science)
- video lectures (Intro & Widgets, Libraries, ATLAS Dijet), hands-on sessions, and peer-reviewed project work
- next event: December–January
- contact:
  - Ross Church: ross@astro.lu.se
  - Caterina Doglioni: caterina.doglioni@hep.lu.se
  - Mikael Lund: mikael.lund@teokem.lu.se

In [36]:
```
from rdkit import Chem
from rdkit.Chem.Draw import IPythonConsole
m1 = Chem.MolFromSmiles('n1c2C(=O)NC(N)=Nc2ncc1CNc3ccc(cc3)C(=O)N[C@H](C(O)=O)CCC(O)=O')
m1
```

Out[36]:



In [37]:
```
from rdkit.Chem import Draw
Draw.MolToFile(m1,'aux/folate.svg')
```

In [38]:
```
m1.GetNumAtoms()
```

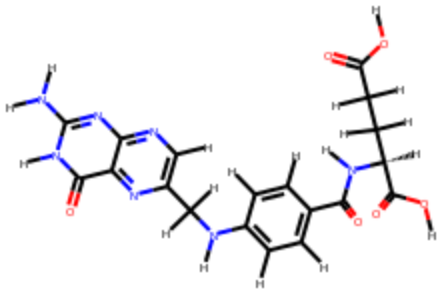
Out[38]: 32

In [39]:
```
Chem.MolToSmiles(m1)
```

Out[39]:
```
'Nc1nc2ncc(CNc3ccc(C(=O)N[C@@H](CCC(=O)O)C(=O)O)cc3)nc2c(=O)[nH]1'
```

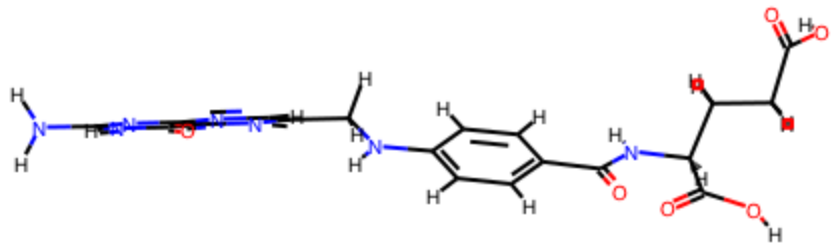
In [40]:
```
m2 = Chem.AddHs(m1) # add hydrogens
m2
```

Out[40]:



In [41]:
```
from rdkit.Chem import AllChem
Chem.AllChem.EmbedMolecule(m2) # make it 3D using ETKDG method
m2
```

Out[41]:



In [42]:
```
import nglview as nv
view = nv.show_rdkit(m2)
view
```

In [43]:
```
print(Chem.MolToMolBlock(m2),file=open('aux/folate.mol','w+'))
```

In [44]:
```
view = nv.show_file('aux/folate.mol')
view
```

In [45]:
```
import mdtraj as md
s = md.load('aux/4mqj.pdb')
print('Number of atoms:', s.n_atoms)
print('Number of residues:', s.n_residues)
chains = [chain for chain in s.top.chains]
n_chains = len(chains)
print('Number of chains:', n_chains)
s14 = s.atom_slice(s.top.select('all and chainid < 4'))
print('Radius of gyration:',md.compute_rg(s14)[0],'nm')
```

```
Number of atoms: 10369
Number of residues: 2386
Number of chains: 24
Radius of gyration: 2.3107479678330973 nm
```

In [46]:
```
import nglview as nv
view = nv.show_pdbid('4mqj')
view
```

In [47]:
```
import matplotlib as mpl
view = nv.show_mdtraj(s)
view.clear_representations(component=0)
for i in range(4):
    chain = [a.index for a in s.top.chain(i).atoms]
    view.add_representation('spacefill', selection=chain, color=mpl.colors.to_hex(plt.cm.tab20(i)))
view
```

In [48]:
```
def viewColorScheme(molecule,dataframe):
    dataframe = dataframe.copy()
    dataframe['3-Letter  Symbol'] = dataframe['3-Letter  Symbol'].str.upper()
    dataframe.set_index('3-Letter  Symbol', drop=True, inplace=True)
    dd = dataframe.dropna()
    dataframe = dataframe.fillna(0)
    preg = (dd['pKx3']-dd['pKx3'].min()) / (dd['pKx3'].max() - dd['pKx3'].min())
    colorscheme = pd.Series([mpl.colors.to_hex(c) for c in plt.cm.rainbow_r(preg)],index=preg.index)
    view = nv.show_mdtraj(molecule)
    view.clear_representations(component=0)
    for res in [res for chain in chains[:4] for res in chain.residues ]:
        atoms = [a.index for a in res.atoms]
        if dataframe.loc[res.name]['pKx3'] == 0:
            view.add_spacefill(selection=atoms, color='#ffffff')
        else:
            view.add_spacefill(selection=atoms, color=colorscheme[res.name])
    view.camera = 'orthographic'
    return view
```

In [49]:
```
viewColorScheme(s,df)
```

How to get started

The installation is simple and quick!

- Install miniconda
- miniconda is the light version of anaconda, a package manager that runs on Windows, Mac and Linux

On Mac or Linux

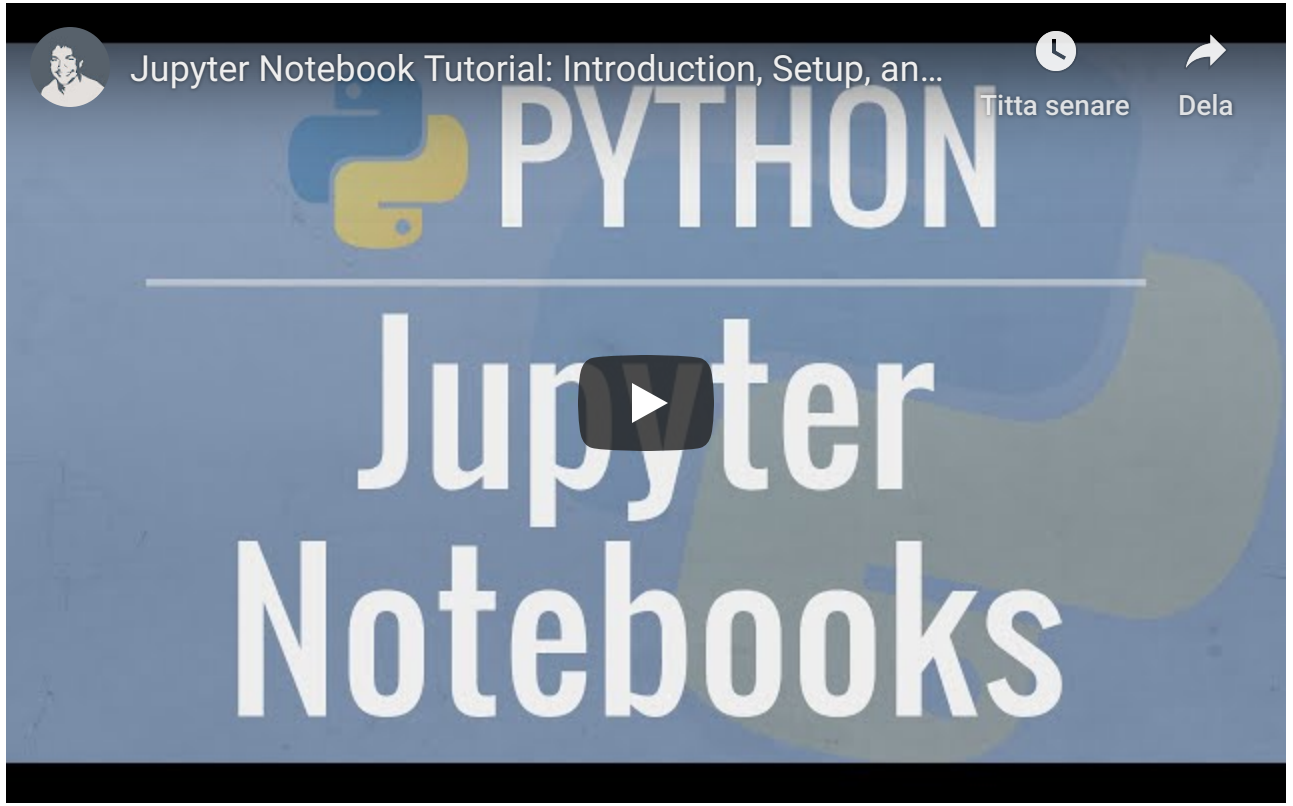
- Download the installation script for your operating system
  - using the terminal: `curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh`
- Install by running the script:
  - type and enter `bash Miniconda3-latest-MacOSX-x86_64.sh`
- Create a `conda` environment with Python 3.7 (the latest version):
  - type and enter `conda create -n myenv python`, myenv is the name of the environment (any name works)
- Activate the environment:
  - `source activate myenv`
- Install notebook, numpy, pandas, matplotlib, scipy:
  - `conda install notebook numpy pandas matplotlib scipy`
- Install RDkit, mdtraj, nglview, ipywidgets
  - we need to specify the channel: `conda install -c conda-forge rdkit mdtraj nglview ipywidgets`
- launch Jupyter notebook: `jupyter-notebook`

On Windows

- Download the installation executable for your operating system
- Install by running the `.exe` file
- Create a new `conda` environment with Python 3.7 (the latest version):
  - open the anaconda prompt from the start menu and navigate to the folder where the course material has been unzipped (e.g. using cd to change directory and dir to list files in a folder)
  - type: `conda create -n myenv python`, myenv is the name of the environment (any name works)
- Activate the environment:
  - `activate myenv`
- Install notebook, numpy, pandas, matplotlib, scipy:
  - `conda install notebook numpy pandas matplotlib scipy`
- Install RDkit, mdtraj, nglview
  - we need to specify the channel: `conda install -c conda-forge rdkit mdtraj nglview ipywidgets`
- launch Jupyter notebook: `jupyter-notebook`

In [50]:
```
from IPython.display import IFrame
IFrame(src='https://www.youtube.com/embed/HW29067qVWk', width=640, height=400)
```

Out[50]:



How to share my notebooks to help other scientists to reproduce my analyses

- Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks

- Saved as an HTML file and provided as Supporting Information

- It is important to provide the list of packages needed to run the notebook:
  - create a conda environment for every project
  - export the conda environment to a yml file: `conda env export > environment.yml`
  - other scientists can quickly reproduce your environment: `conda env create -f environment.yml`

- `notebook.ipynb` + data + `environment.yml` in a zip file as Supporting Information

- Example: refnx: neutron and X-ray reflectometry analysis in Python

How to share my notebooks to help other scientists to reproduce my analyses

- Create a GitHub repository
- Upload your notebook and `environment.yml`
- myBinder allows you to run the notebook in the repository on a server: no need to download and install
- Example: refnx: neutron and X-ray reflectometry analysis in Python