Introduction

Scientific Programming in Python

About the course

Times:

- One **session** every *Tuesday 12:00 - 14:00*
- One **session** every *Thursday 12:00 - 14:00*
- No mandatory attendance
- Lecture will be filmed

Approach:

- Lectures will be a mix of concepts, coding tutorial and interactive exercises
- This year we try to do less talking and let you do more coding

About the course

Grading & Credits

- Weekly homework, done individually
- Homework corrected automatically, and you get the tests, too!
- 12 homework-sheets plus bonus exercises once in a while
- Pass **10** (normal + bonus) to get the Schein
- 4-ECTS Schein for the “Profilbildender Wahlbereich”
    - Everybody who wants to switch to the new exam regulations can make this course count for the new “*Methods of Cognitive Science*” module
- After passing the **10** homeworks you **can** take the final exam and get a grade (helpful for the new module)
- Everybody who passed enough homeworks will at least get a “passed”

About us

Rüdiger

- 1st semester PhD
- Working at AIM/inserve
- 4 years of Python experience
- rbusche@uos.de

Chris

- 5th Master
- 4 years of Python experience
- cstenkamp@uos.de

- If you have any questions, don't hesitate to write us an email or in the forum!
- If you have suggestions for content, please also write!
- If you encounter errors in the the slides or the homework, please do so via a github-issue! The repository for homework is https://github.com/scientificprogrammingUOS

This lecture is for you!

- Last year’s lecture was recorded and slides are uploaded upfront...
- Actively participate
- Ask question when you don’t understand something, everything else is a waste of your time

Why scientific programming?

Science

- Build and organize knowledge
- Test explanation about our world
- Communicate our results to others
- Systematically
- Objectively
- Transparently
- Reproducibly

Otherwise it’s not science.

Cf. PyData Ann Arbor: Katy Huff | Doing Our Best: Practices in Open, Reproducible, Scientific Computing

Programming helps us

- Build and organize knowledge → by building databases of scientific results
- Test explanation about our world → by automating experiments
- Communicate our results to others →by sharing code on top of papers
- Systematically → by easily making analyses exhaustive
- Objectively → computers are not subject to human biases (computer programs still are)
- Transparently → by using open source tools and sharing our analyses
- Reproducibly → by codifying our analyses we make them reproducible

You need to write **clean code**!

Cf. PyData Ann Arbor: Katy Huff | Doing Our Best: Practices in Open, Reproducible, Scientific Computing

Why Python?

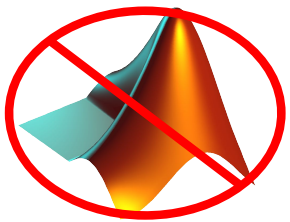
Python

- Create by Guido van Rossum in the 90s
- Now open source project developed by the Python Software foundation
- High-level language (no hardware-knowledge necessary)
- Interpreted and dynamically typed language
- Consistent and minimal syntax
- → easy to learn and write
- Great ecosystem and great community!

[1]

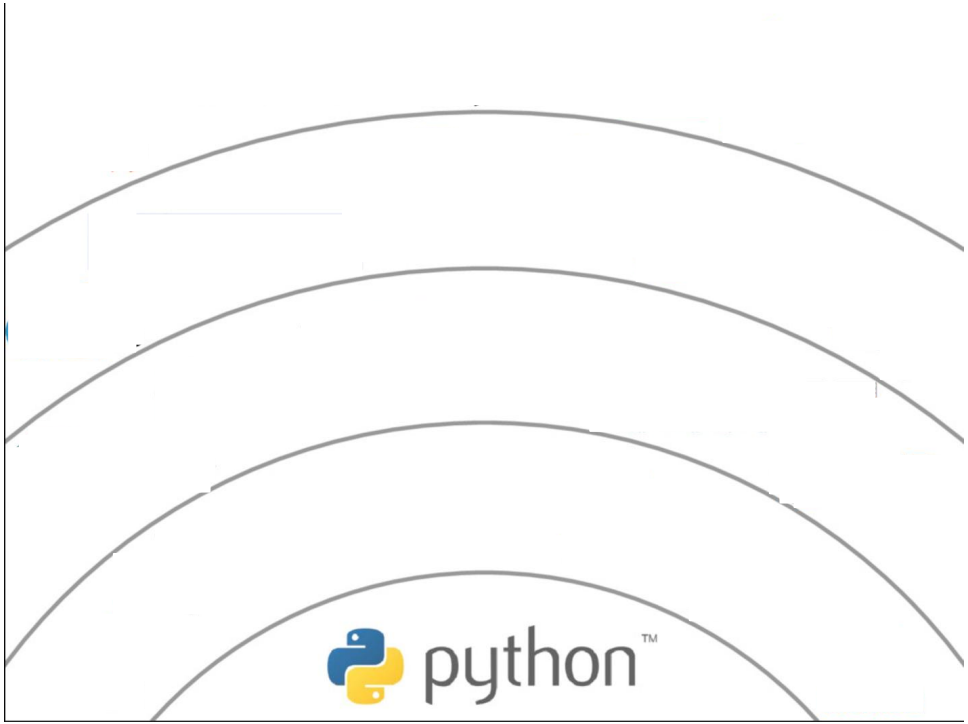
Python is better

- Better than Matlab → As a free and open source project you can save money and actually share your results
- Better than Java → Get more done with less code and without overly complex object orientation.
- Better than C++ (at least for science) → With a great ecosystem and a great community you can get stuff done, instead of trying to figure out documentation yourself
- Better than R →As a general purpose programming language you can do anything with Python not just statistics
- Better than Julia →Python is more versatile and mature. Julia is interesting if you write algorithms



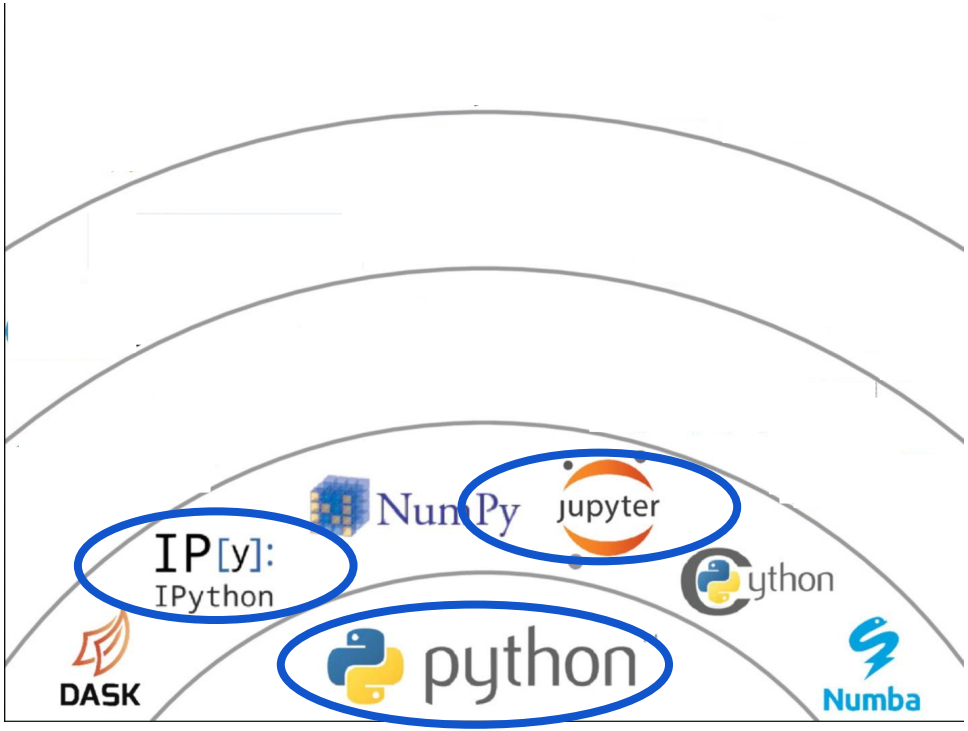
[1,2,3,4,5,6]

Our path through scientific python



https://www.datacamp.com/community/blog/python-scientific-computing-case

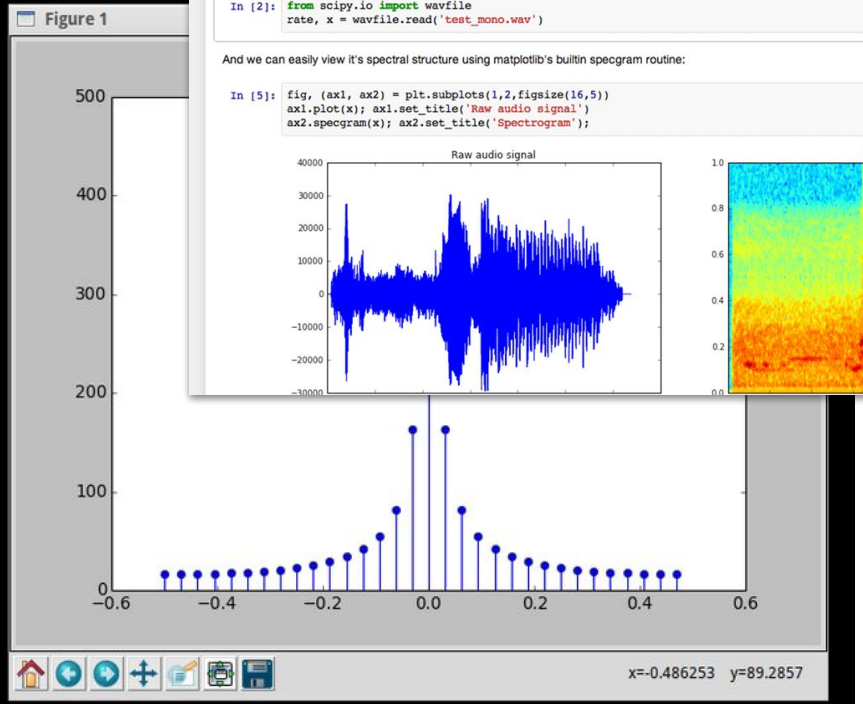
Our path through scientific python

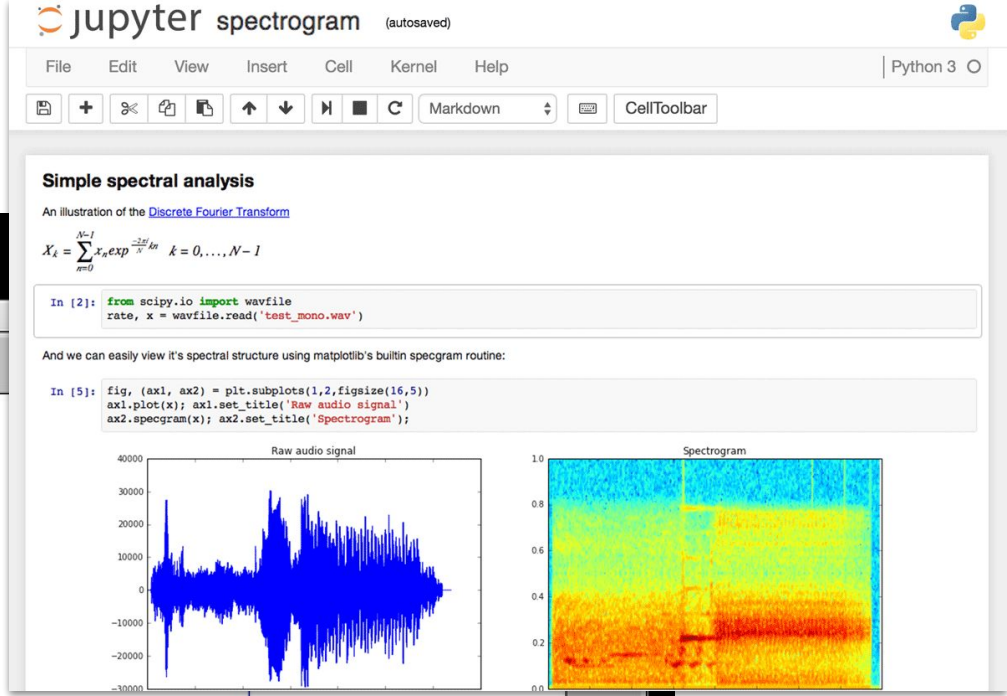


https://www.datacamp.com/community/blog/python-scientific-computing-case

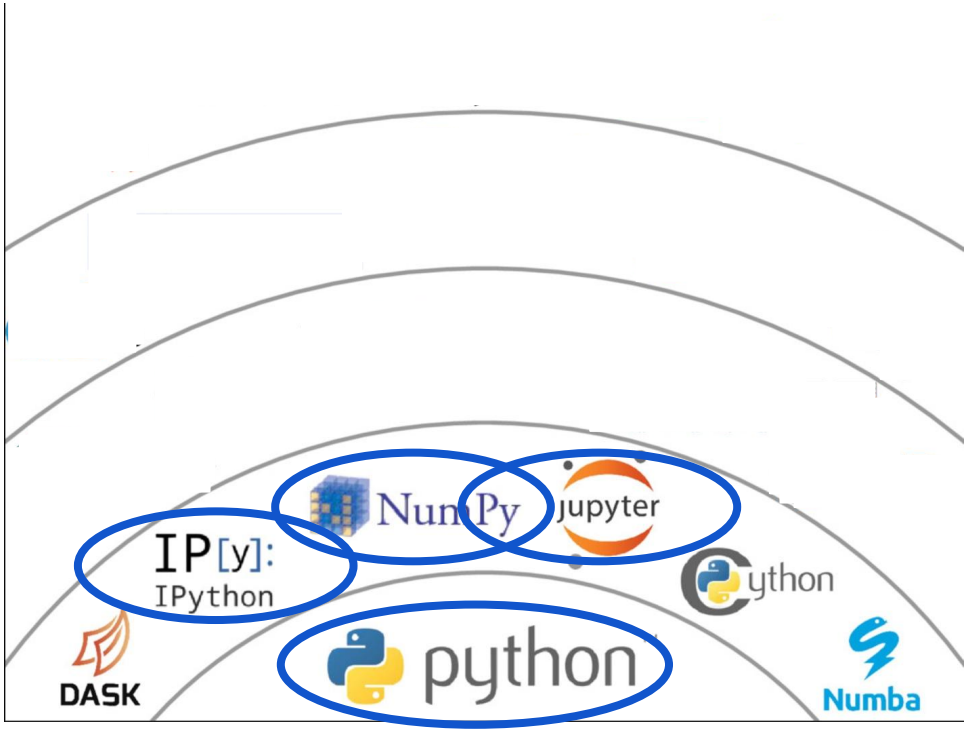
Python, Jupyter, IPython

```
Python 3.2.3 (default, Sep 25 2013, 18:25:56)
Type "copyright", "credits" or "license" for more information.

IPython 1.1.0 -- 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.
Using matplotlib backend: TkAgg

In [1]: from numpy.fft import *

In [2]: a = arange(32)

In [3]: A = fft(a)

In [4]: f = fftfreq(32)

In [5]: stem(f,abs(A))
Out[5]: <Container object of 3 artists>

In [6]:
```





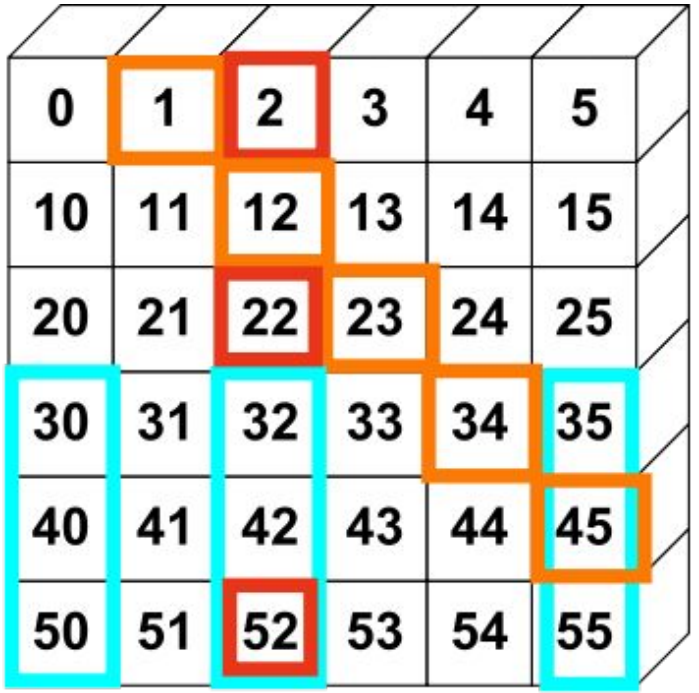
Our path through scientific python



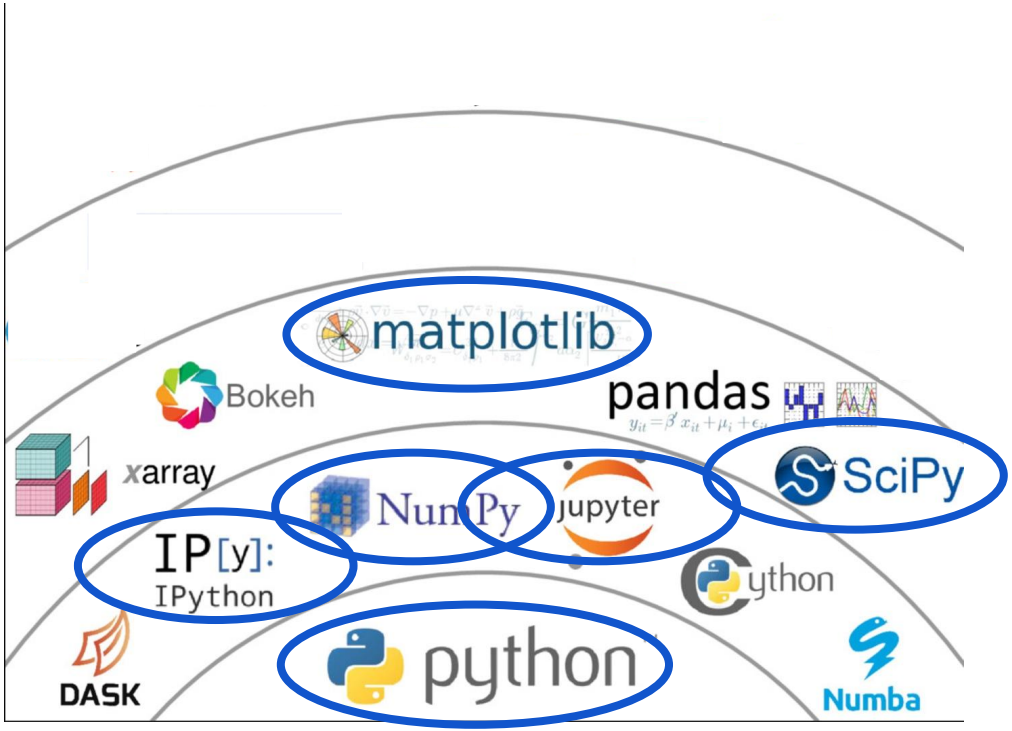
https://www.datacamp.com/community/blog/python-scientific-computing-case

NumPy

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])

>>> a[3:,[0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45]])
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1],
                 dtype=bool)
>>> a[mask,2]
array([2,22,52])
```

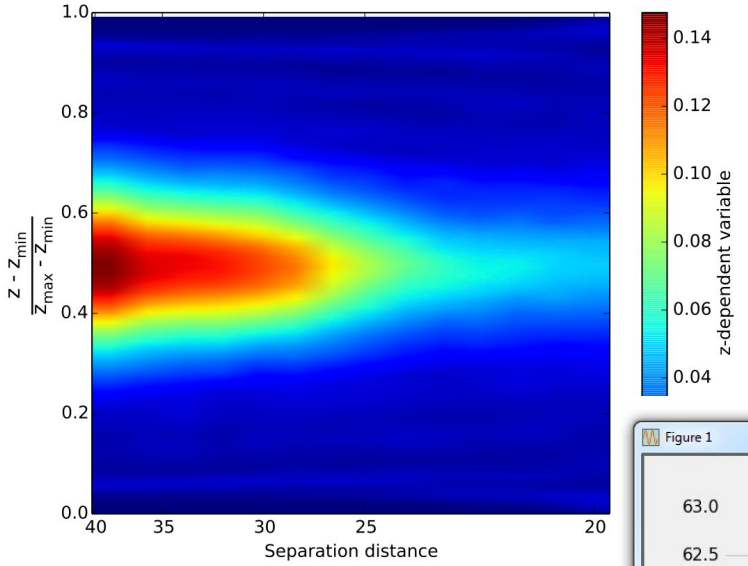


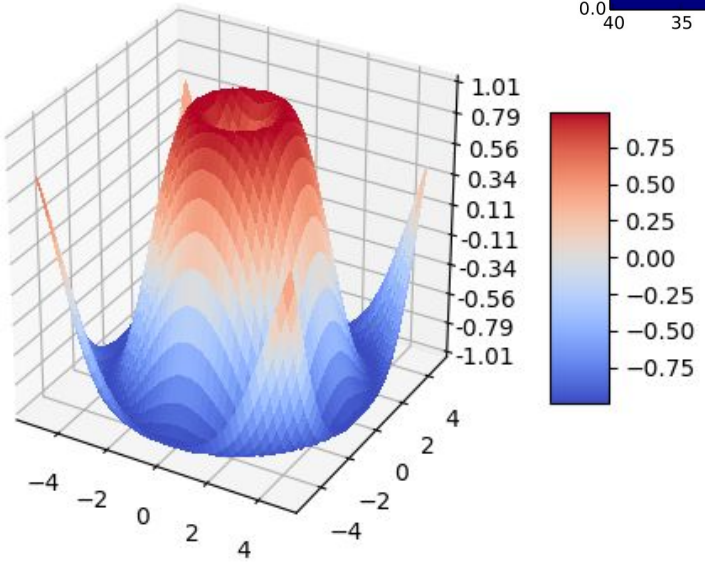
Our path through scientific python



https://www.datacamp.com/community/blog/python-scientific-computing-case

Matplotlib


1.0
0.8
0.6
0.4
0.2
0.0
$\frac{z - z_{min}}{z_{max} - z_{min}}$
40
35
30
25
20
Separation distance
0.14
0.12
0.10
0.08
0.06
0.04
z-dependent variable


1.01
0.79
0.56
0.34
0.11
-0.11
-0.34
-0.56
-0.79
-1.01
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−4
−2
0
2
4
−4
−2
0
2
4

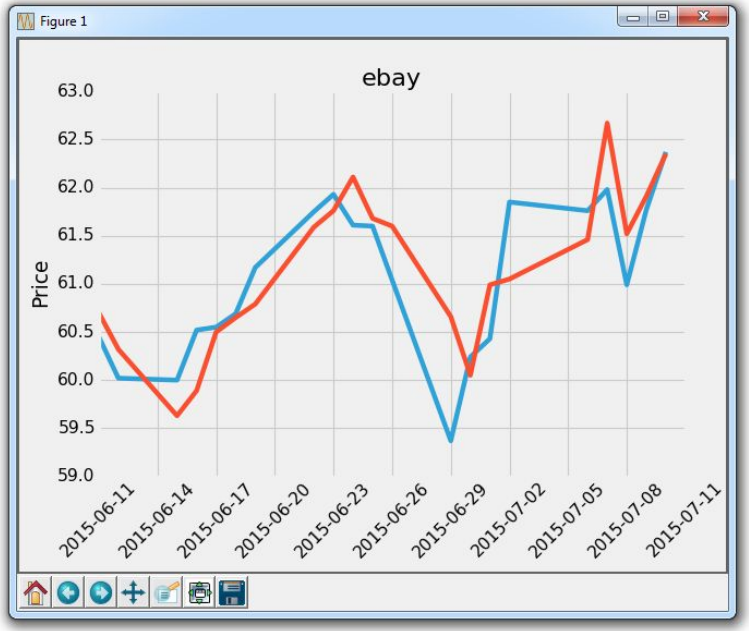
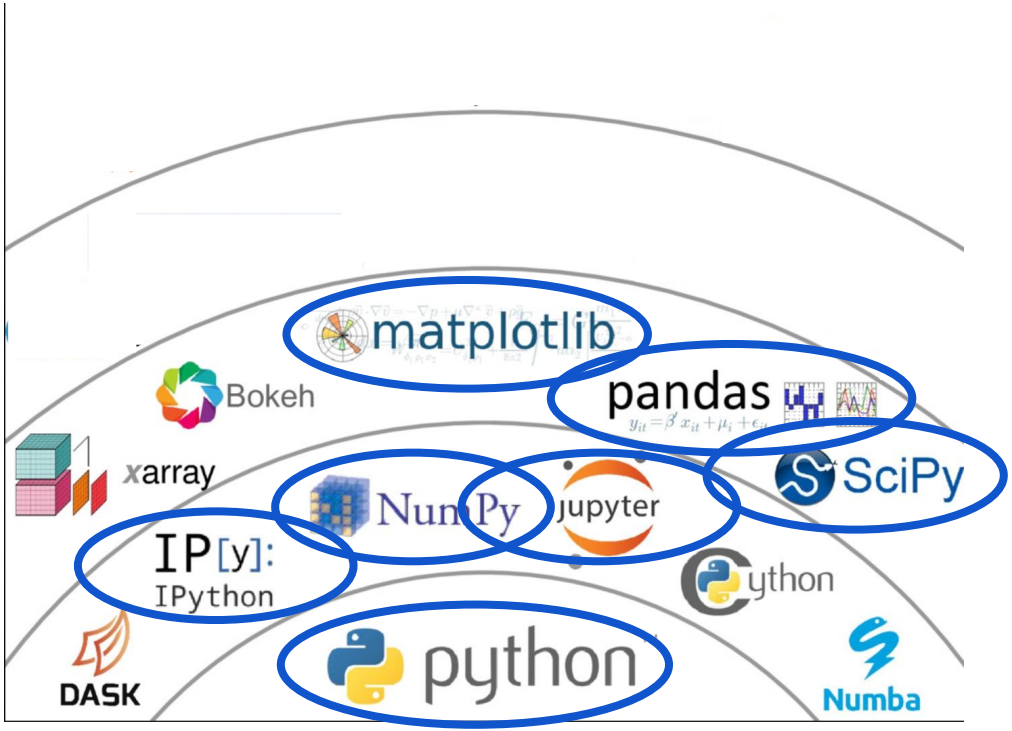


Figure 1
ebay
63.0
62.5
62.0
61.5
61.0
60.5
60.0
59.5
59.0
Price
2015-06-11
2015-06-14
2015-06-17
2015-06-20
2015-06-23
2015-06-26
2015-06-29
2015-07-02
2015-07-05
2015-07-08
2015-07-11

Our path through scientific python



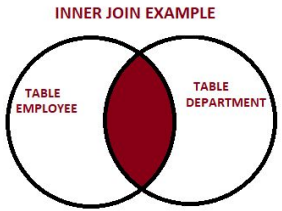
https://www.datacamp.com/community/blog/python-scientific-computing-case

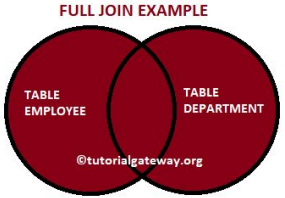
Pandas

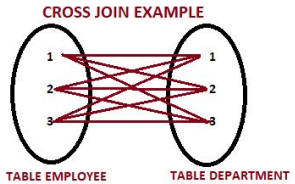
```
# Create a dataframe with dates as your index
States = ['NY', 'NY', 'NY', 'NY', 'FL', 'FL', 'GA', 'GA', 'FL', 'FL']
data = [1.0, 2, 3, 4, 5, 6, 7, 8, 9, 10]
idx = pd.date_range('1/1/2012', periods=10, freq='MS')
df1 = pd.DataFrame(data, index=idx, columns=['Revenue'])
df1['State'] = States

# Create a second dataframe
data2 = [10.0, 10.0, 9, 9, 8, 8, 7, 7, 6, 6]
idx2 = pd.date_range('1/1/2013', periods=10, freq='MS')
df2 = pd.DataFrame(data2, index=idx2, columns=['Revenue'])
df2['State'] = States
```

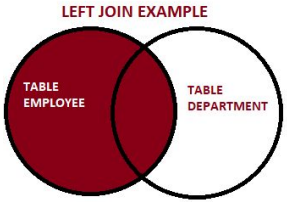
```
# Combine dataframes
df = pd.concat([df1,df2])
df
```

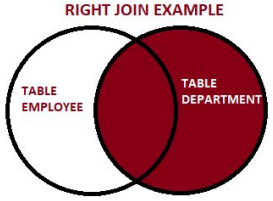
	Revenue	State
2012-01-01	1.0	NY
2012-02-01	2.0	NY
2012-03-01	3.0	NY
2012-04-01	4.0	NY
2012-05-01	5.0	FL
2012-06-01	6.0	FL
2012-07-01	7.0	GA
2012-08-01	8.0	GA

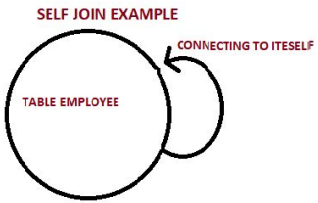




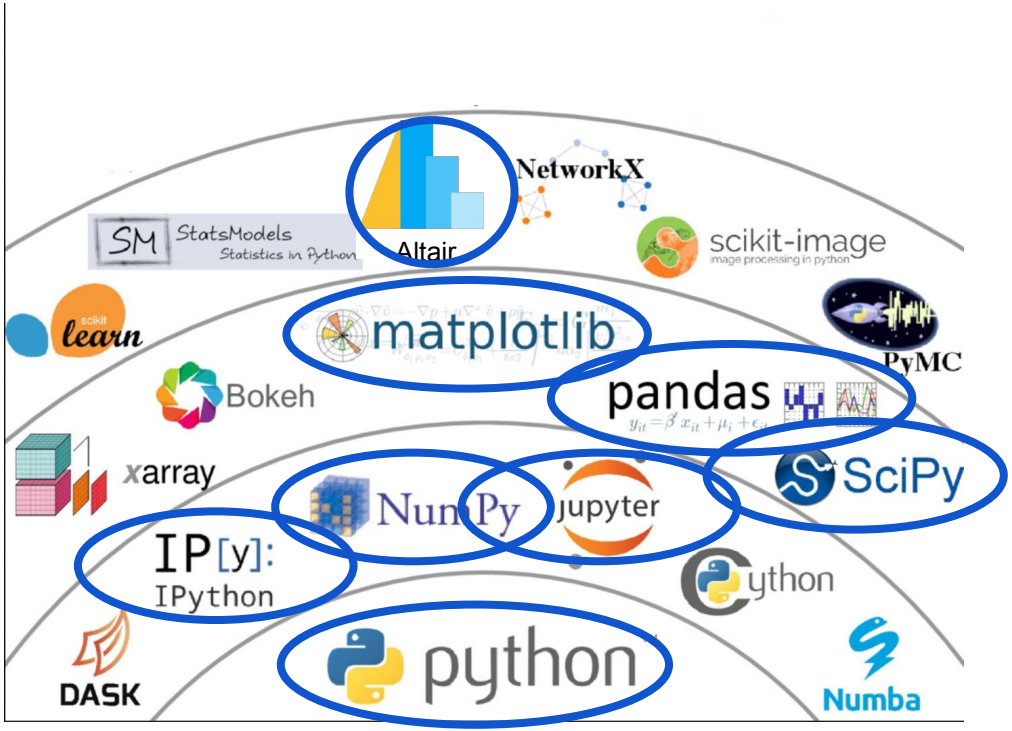






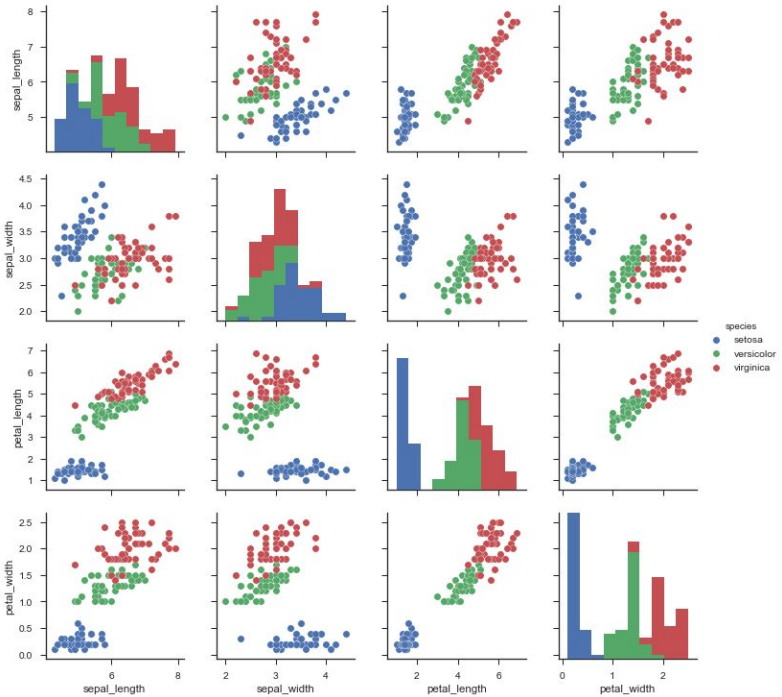


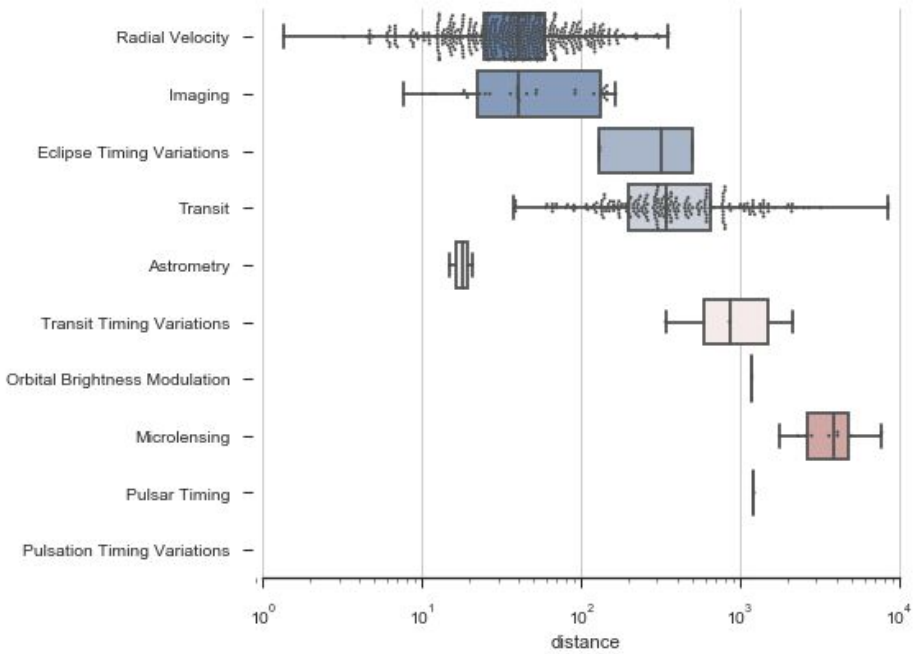
Our path through scientific python



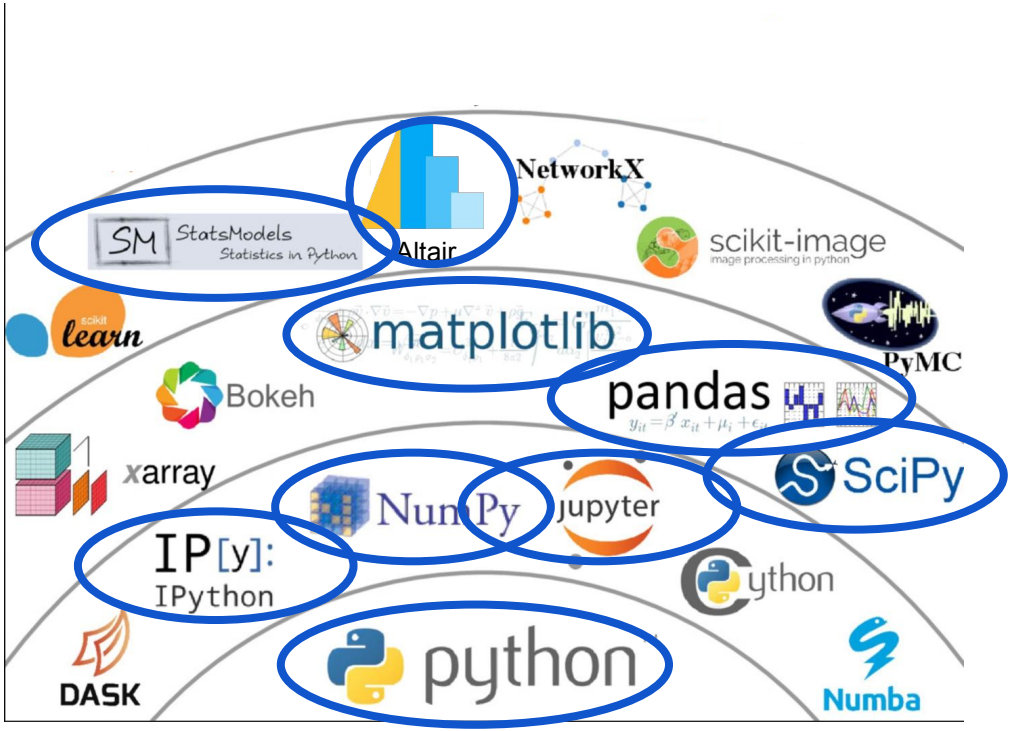
https://www.datacamp.com/community/blog/python-scientific-computing-case

Statistical visualization





Our path through scientific python



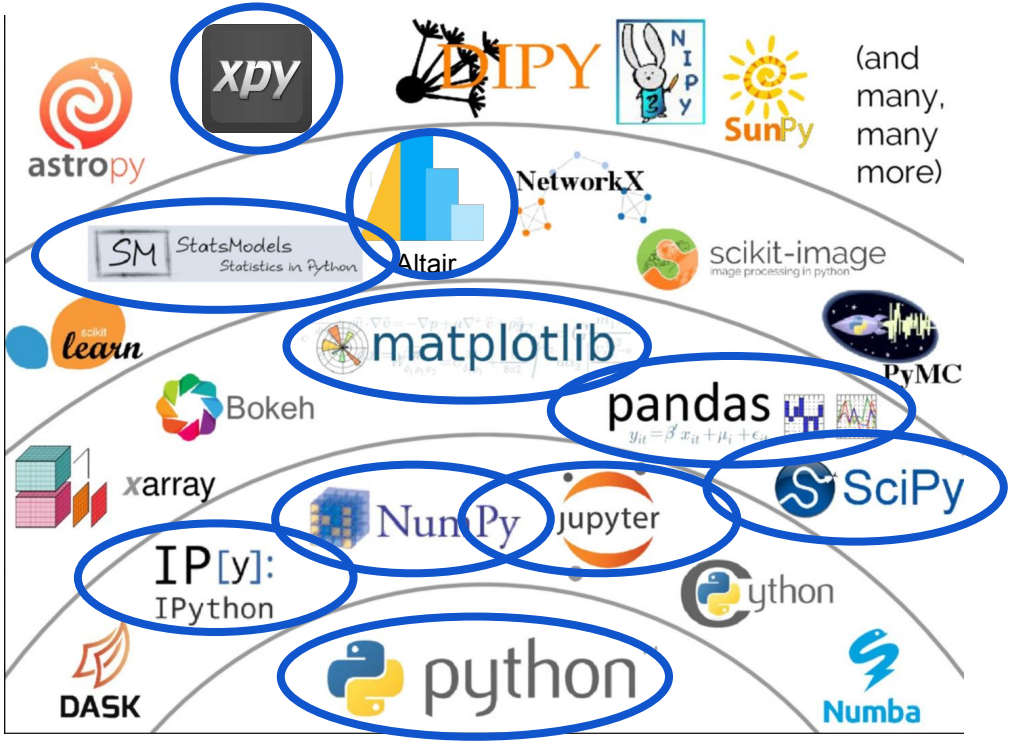
https://www.datacamp.com/community/blog/python-scientific-computing-case

Statsmodels

```
In [5]: results = smf.ols('Lottery ~ Literacy + np.log(Pop1831)', data=dat).fit()
```

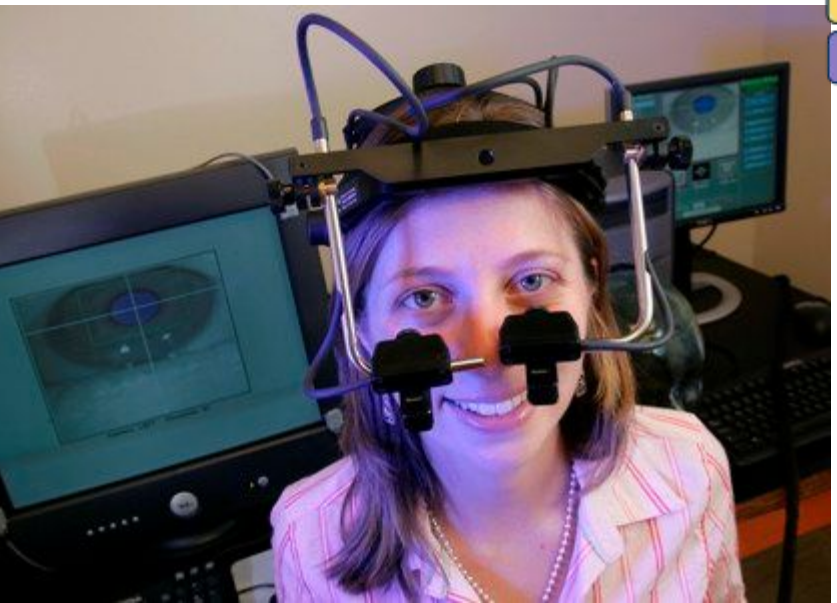
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Lottery   R-squared:                       0.348
Model:                            OLS   Adj. R-squared:                  0.333
Method:                 Least Squares   F-statistic:                     22.20
Date:                Tue, 28 Feb 2017   Prob (F-statistic):           1.90e-08
Time:                        21:38:05   Log-Likelihood:                -379.82
No. Observations:                  86   AIC:                             765.6
Df Residuals:                      83   BIC:                             773.0
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    246.4341     35.233      6.995      0.000     176.358     316.510
Literacy      -0.4889      0.128     -3.832      0.000      -0.743      -0.235
np.log(Pop1831)   -31.3114      5.977     -5.239      0.000     -43.199     -19.424
==============================================================================
Omnibus:                        3.713   Durbin-Watson:                   2.019
Prob(Omnibus):                  0.156   Jarque-Bera (JB):                3.394
Skew:                          -0.487   Prob(JB):                        0.183
Kurtosis:                       3.003   Cond. No.                         702.
==============================================================================
```

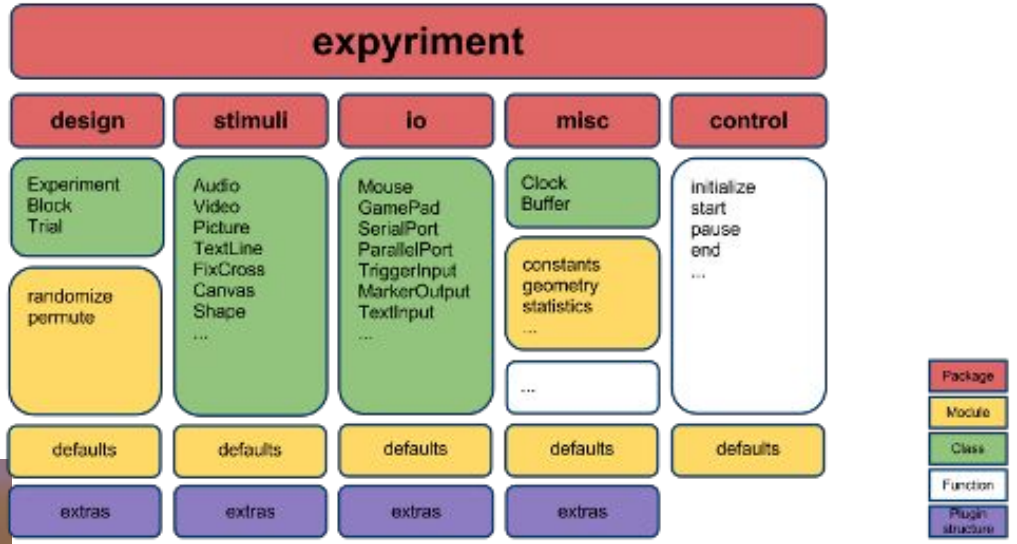
Our path through scientific python



https://www.datacamp.com/community/blog/python-scientific-computing-case

Expyriment





http://docs.expyriment.org/Overview.html

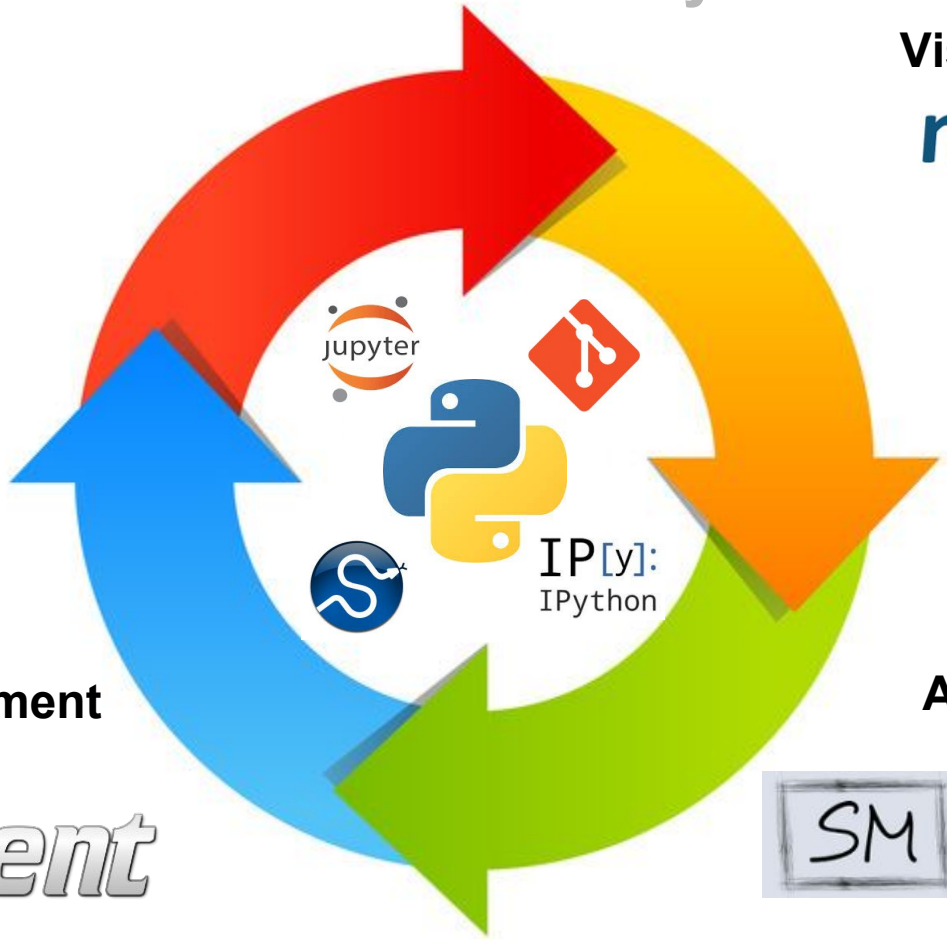
Your workflow with Python

Extracting your data

NumPy

pandas

$$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Visualizing your data

matplotlib

plotnine

Making your experiment

expyriment

Analyzing your data

SM StatsModels
Statistics in Python

Let’s compute!

Go to https://bit.ly/2uL84ux and click “launch binder”

Outline

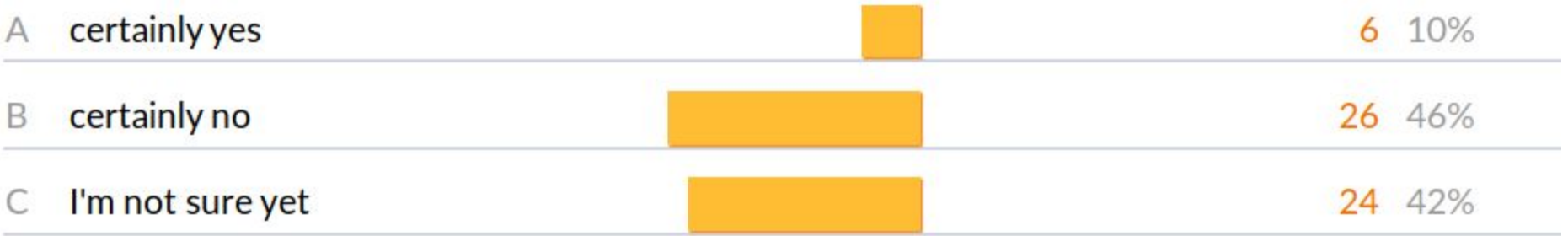
1. Intro & Organization
2. Basic Python
3. Advanced Python
4. Numerical Computing with NumPy
5. Visualizations with Matplotlib
6. Framing your Data with Pandas
7. Cleaning Data with Pandas
8. Analyzing with Pandas
9. Creating Experiments with Expyriment
10. Statistical Visualization with ggplot
11. Statistical Modeling with statsmodels
12. Interactive Data Analysis with Altair and Jupyter Widgets
13. Performance Optimization

Basic Programming in Python:

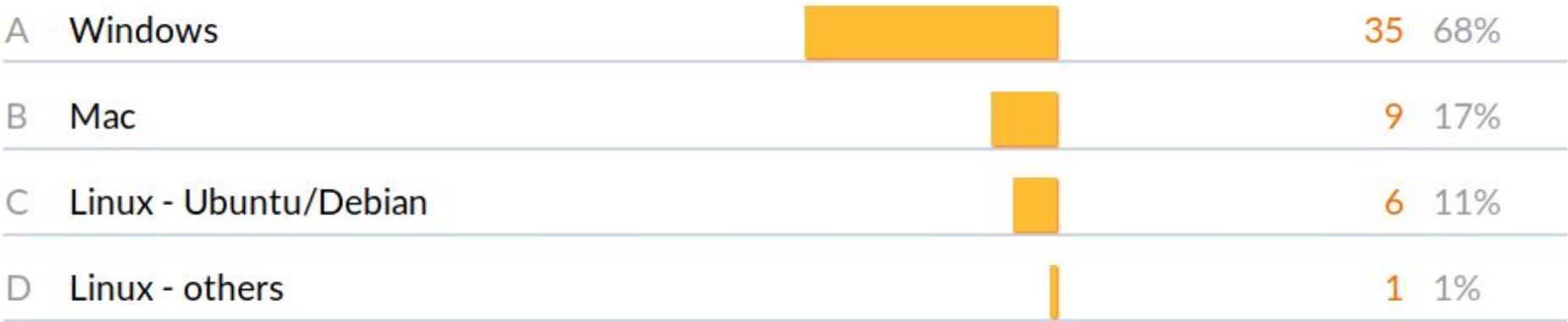
Structure

- **Week 1: Introduction**
- Week 2: Syntax & Variables
- Week 3: Control Structures
- Week 4: Lists & Collections
- Week 5: RegEx & Strings
- Week 6: Sorting & I/O
- Week 7:Debugging, Errors & Strategies
- Week 8: Python Packages
- Week 9: Practical Python & Good practices
- Week 10: Object Oriented Programming
- Week 11: Time, Space and documentation
- Week 12: Numpy & Matplotlib
- Week 13: Outlook & wrapping up
- Week 14: TBA

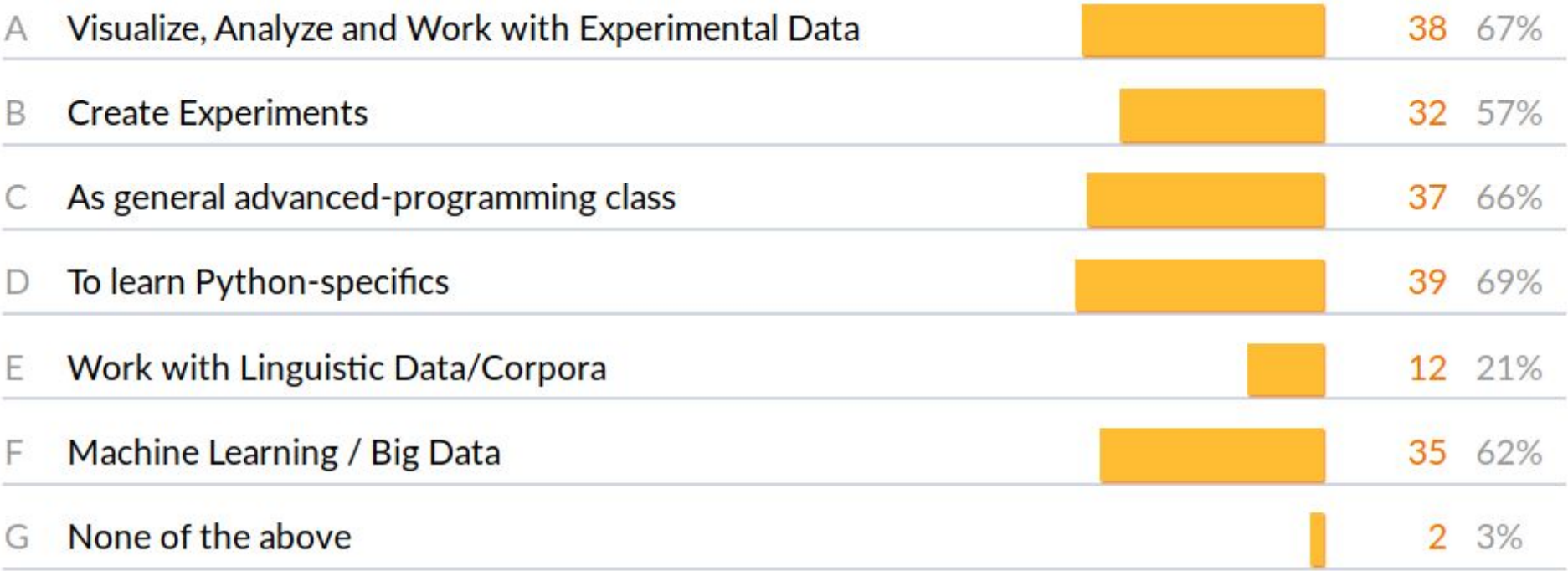
Do you want to take the exam and get a grade for this class?



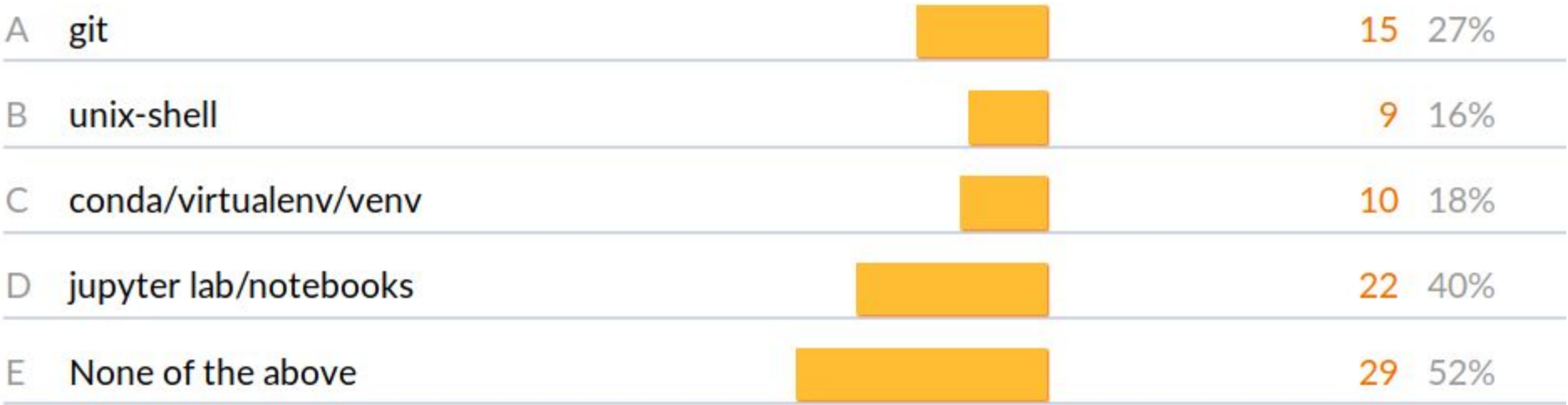
Which Operating System do you use?



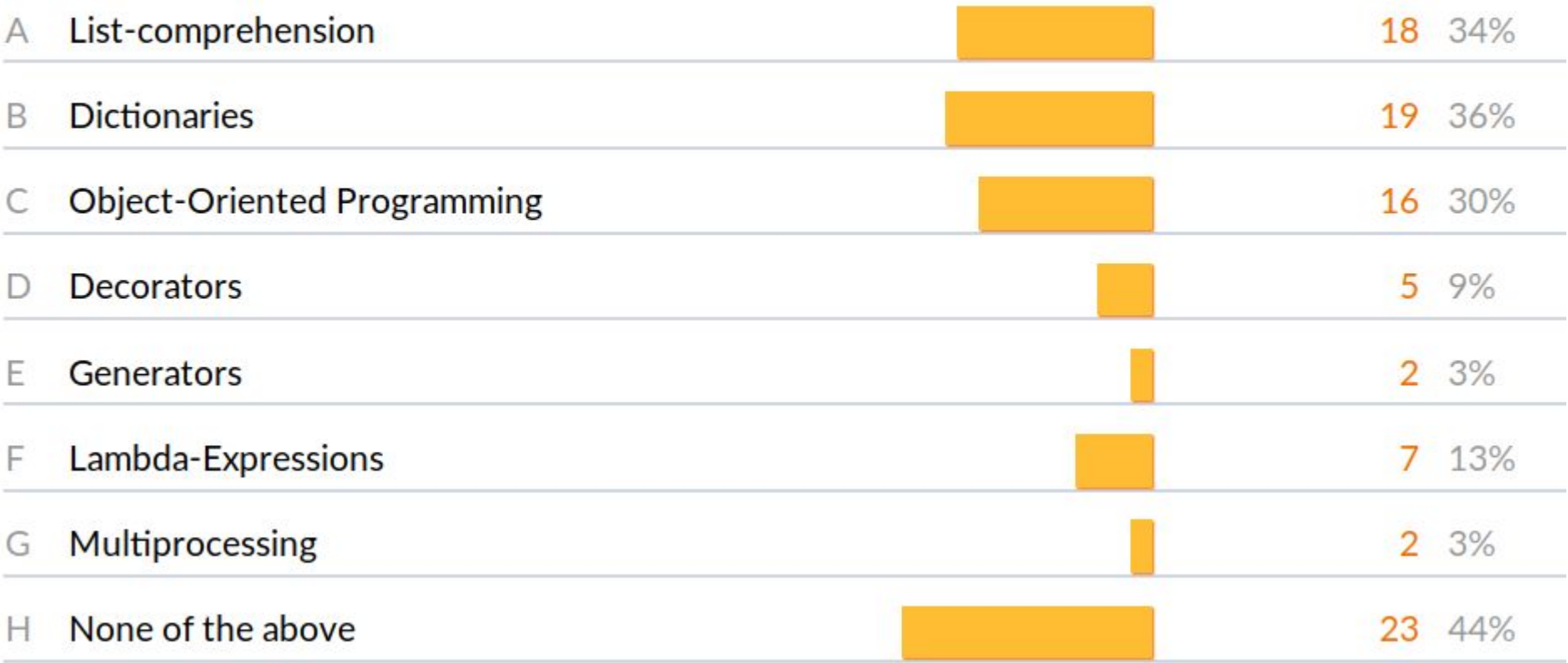
What do you want to learn to code for?



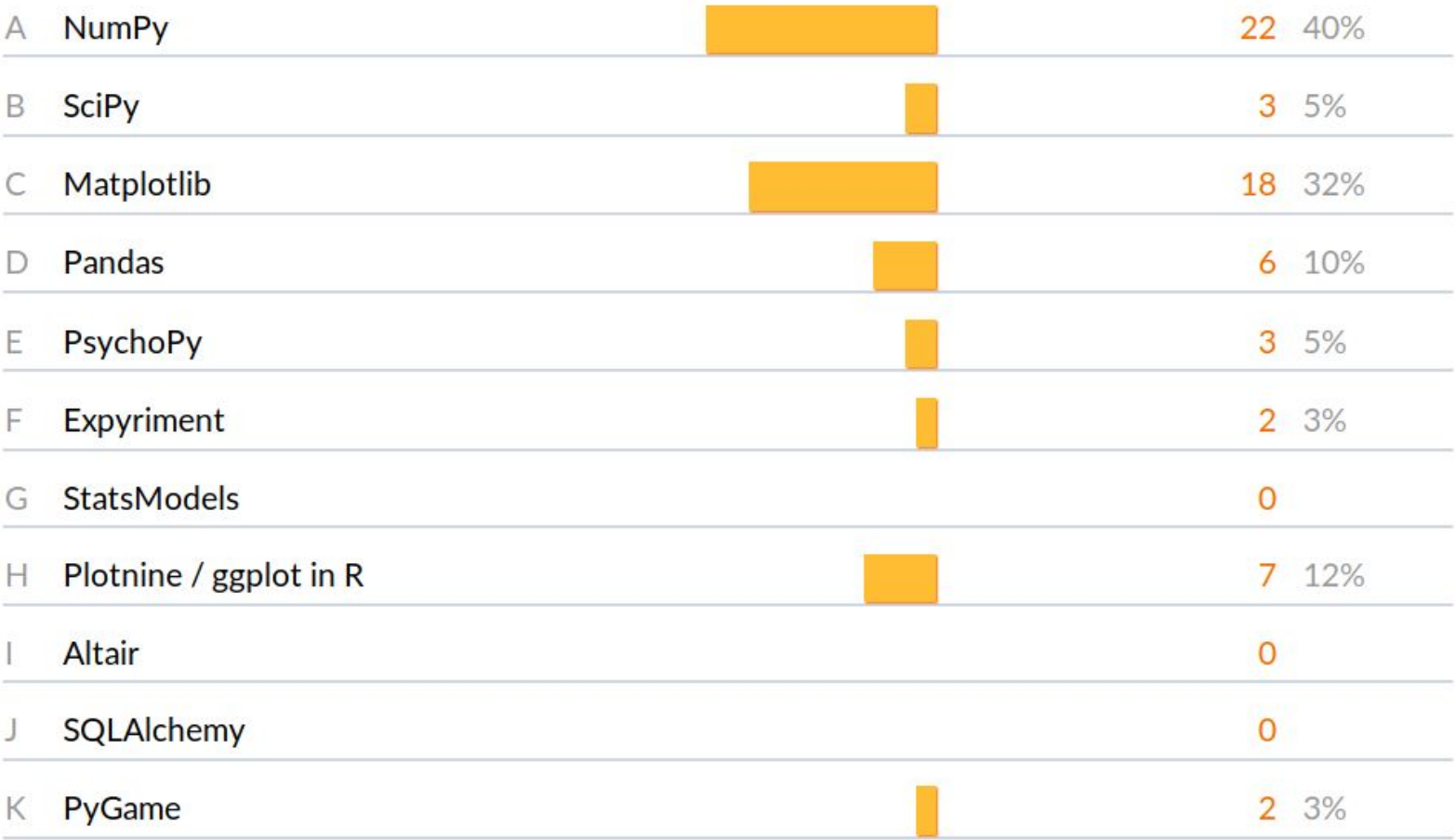
Do you have basic knowledge of...



Which Python-Concepts did you use so far?



What libraries have you used so far?



Setup for the course

Which IDE?

Save yourself the pain and don’t code in notepad, without any kind of syntax highlighting or code completion

- There are as many IDEs as there are opinions about them
- Pycharm, vscode, atom, notepad++, vi, emacs…
- For this course we will only use **JupyterLab**
- Contains all you need and is great for interactive development as usually encountered in scientific contexts

Which IDEs are there?

- Atom https://atom.io/ (or for linux *apt-get install atom*)
  - Useful Packages: **Hydrogen** by *nteract* & **hydrogen-launcher** by *lgeiger,* providing an interactive Kernel
  - Free and open source
  - Recommended for smaller projects (containing only few files)
- Pycharm https://www.jetbrains.com/pycharm/
  - Commercial, closed source, however Community edition free and Professional free as student
  - Features many components - debugger, code analysis features, git integration, ...
  - Useful for big projects, not recommended for homework of this course
- vi
  - Integrated into Unix-systems, runs *inside* the terminal
  - Hard to master, but supposedly much faster once you did
- Jupyter Lab
  - Can not only work with pure code-files, but also *Notebook-Files* that contain code, formatted text and results of running your code
    - These notebooks are nicely rendered on Github and can be exported to HTML or PDF, or simply to a .py-script
  - Can also edit standard-python-files and use them with an interactive Kernel

Virtual environments

- Virtual environments are sandboxes for your python and its packages - allowing you to have different Python versions with different packages side by side
- Working with the default Python leads to a mess or can even corrupt your operating system!
- Conda is the easiest option to get Python virtual envs on all platforms
- Cheat sheets:
  - http://know.continuum.io/rs/387-XNW-688/images/conda-cheatsheet.pdf
  - https://conda.io/projects/conda/en/latest/_downloads/1f5ecf5a87b1c1a8aaf5a7ab8a7a0ff7/conda-cheatsheet.pdf

Install Anaconda or Miniconda

- **Anaconda** is a *Python distribution* made for scientific computing, packed with its own package manager (*conda*).
  - Anaconda contains >720 pre-installed packages at ~3GB
  - Download: https://www.anaconda.com/distribution/

- **Miniconda** is the same as Anaconda, just without all the pre-installed packages (besides the package manager)
  - Thus its only 66MB, and every package you need can be installed via *conda*
  - Download: https://docs.conda.io/en/latest/miniconda.html

Installation instructions: Miniconda & Linux

- Download your version from https://docs.conda.io/en/latest/miniconda.html
- *bash Miniconda3-latest-Linux-x86_64.sh* (saying yes when it asks you to add it to the terminal)
  - (*which python* should now answer .../anaconda3/bin/python)
  - (alternatively: *conda update --all*)
- If you don't have git already (try by running *which git*), install it using *sudo apt-get install git*

To have jupyterlab globally on your system:

- *conda install jupyter*
- *conda install jupyterlab*
- *conda install conda-forge::nodejs*
- *jupyter labextension install @lckr/jupyterlab_variableinspector*

To create the environment for the class:

- *git clone https://github.com/scientificprogrammingUOS/lectures.git*
- *conda env create -f lectures/environment.yml*
- *conda activate scientific_programming*
- *jupyter labextension install @lckr/jupyterlab_variableinspector*

Installation instructions: Miniconda & Windows

- Download your version from https://docs.conda.io/en/latest/miniconda.html
- Run the graphical installer.
  - **Make sure to add conda to your PATH**[1], such that you can use it from your standard terminal.
- Afterwards, open the command-prompt as Administrator (hit Win-Key, type “cmd”, right-click “Command Prompt”, select “as Admin”)
  - Test if your installation was correct by running *where python*
    → it should return a path containing .../Anaconda3/…
  - Update your Conda installation by running *conda update conda*
- Install git from https://git-scm.com/downloads
  - Make sure that git **can** be used from your command-line[2]
  - Make sure that you use one of the two options **committing unix-style**[3]
  - Leave everything else the way it is
- Afterwards you should have the commands *python*, *conda* and *git* as registered commands[4]

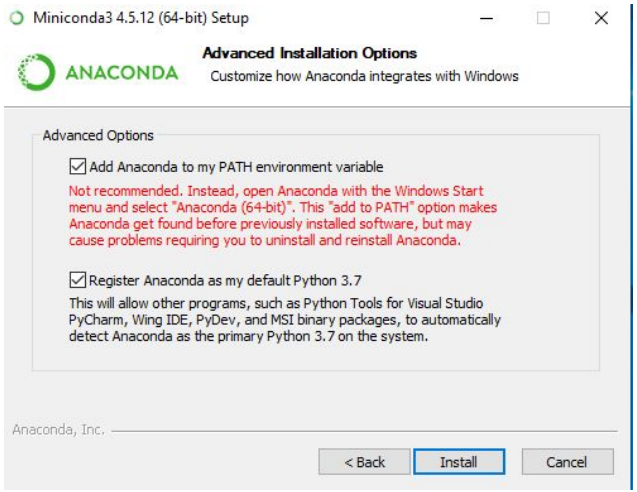
To have jupyterlab globally on your system:

- *conda install jupyter*
- *conda install jupyterlab*
- *conda install conda-forge::nodejs*
- *jupyter labextension install @lckr/jupyterlab_variableinspector*[5]

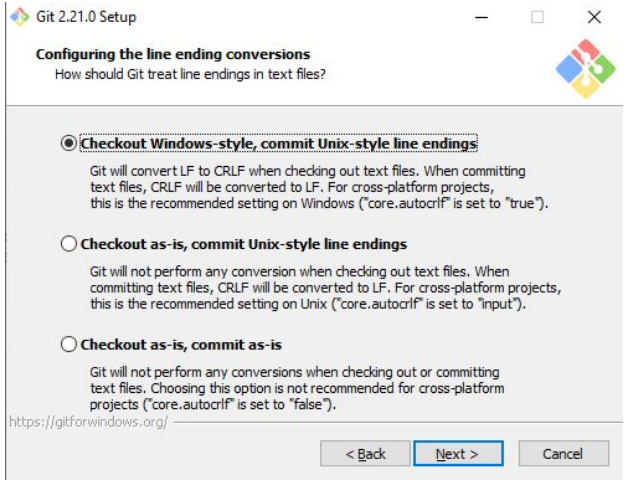
To create the environment for the class:

- *git clone https://github.com/scientificprogrammingUOS/lectures.git*
- *conda env create -f lectures/environment.yml*
- *conda activate scientific_programming*
- *jupyter labextension install @lckr/jupyterlab_variableinspector*[5]

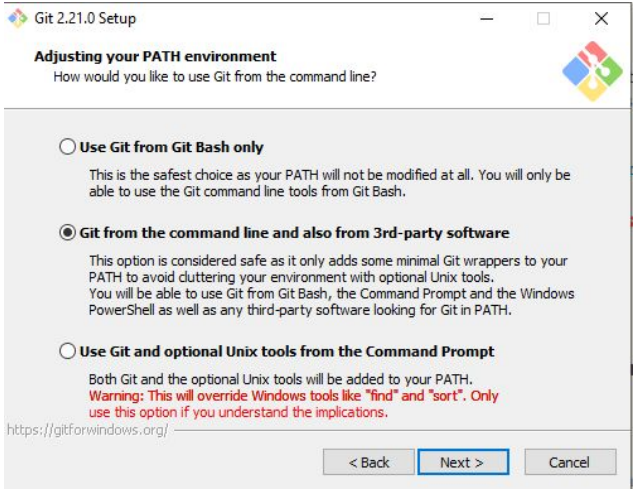
1)


Miniconda3 4.5.12 (64-bit) Setup
ANACONDA
Advanced Installation Options
Customize how Anaconda integrates with Windows
Advanced Options
Add Anaconda to my PATH environment variable
Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.
Register Anaconda as my default Python 3.7
This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.7 on the system.
Anaconda, Inc.
< Back
Install
Cancel

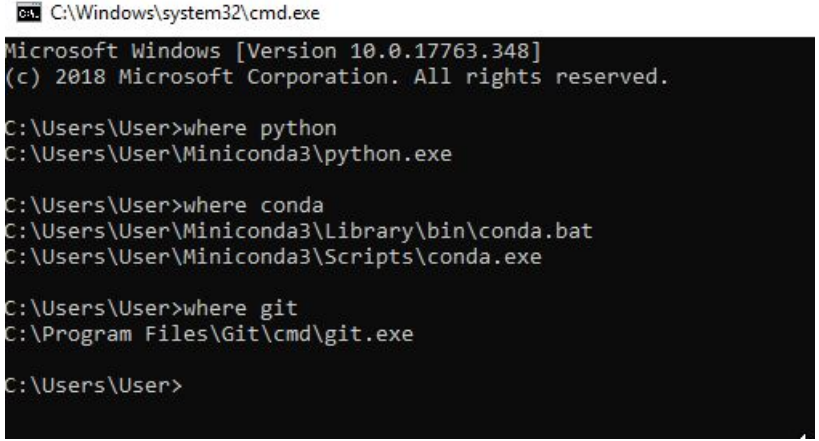
2)


Git 2.21.0 Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").
Checkout as-is, commit Unix-style line endings
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").
Checkout as-is, commit as-is
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").
https://gitforwindows.org/
< Back
Next >
Cancel

3)


Git 2.21.0 Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.
Git from the command line and also from 3rd-party software
This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools.
You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.
Use Git and optional Unix tools from the Command Prompt
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.
https://gitforwindows.org/
< Back
Next >
Cancel

4)


C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.17763.348]
(c) 2018 Microsoft Corporation. All rights reserved.
C:\Users\User>where python
C:\Users\User\Miniconda3\python.exe
C:\Users\User>where conda
C:\Users\User\Miniconda3\Library\bin\conda.bat
C:\Users\User\Miniconda3\Scripts\conda.exe
C:\Users\User>where git
C:\Program Files\Git\cmd\git.exe
C:\Users\User>

After Installation

- Once you activate your environment using *conda activate scientific_python,* your shell should indicate that you're inside this environment
- Note that you have to **activate your this environment every time you work on the exercises!**
- To test if all packages are installed successfully, run *conda list* and check if all demanded packages are indeed listed.
- To start working inside jupyter lab, navigate to the correct directory using *cd,* and then start jupyterlab by executing *jupyter lab .* (yes, including the dot)[5]

5) If the commands involving jupyter don't work on Windows, try using a hyphen instead of a space:
   → *jupyter-labextension install @lckr/jupyterlab_variableinspector*
   → *jupyter-lab .*

Thanks for your attention!

- We will have have a feedback-questionnaire after 4-5 sessions
- Any questions and remarks please via email!
- Content-suggestions are always welcome!

Thanks to Patrick Faion and Brian Lewis for the old *Scientific Programming in Python* lecture

Sources

1. https://commons.wikimedia.org/wiki/File:Python.svg
2. https://pixabay.com/vectors/swiss-army-knife-pocket-knife-blade-154314/
3. https://en.wikipedia.org/wiki/R_(programming_language)
4. https://commons.wikimedia.org/wiki/File:Matlab_Logo.png
5. https://commons.wikimedia.org/wiki/File:Images_200px-ISO_C%2B%2B_Logo_svg.png
6. https://pt.wikipedia.org/wiki/Julia_(linguagem_de_programa%C3%A7%C3%A3o)
7. https://commons.wikimedia.org/wiki/File:Git_icon.svg
8. https://farm2.staticflickr.com/1482/24588096069_59a0513790_z.jpg