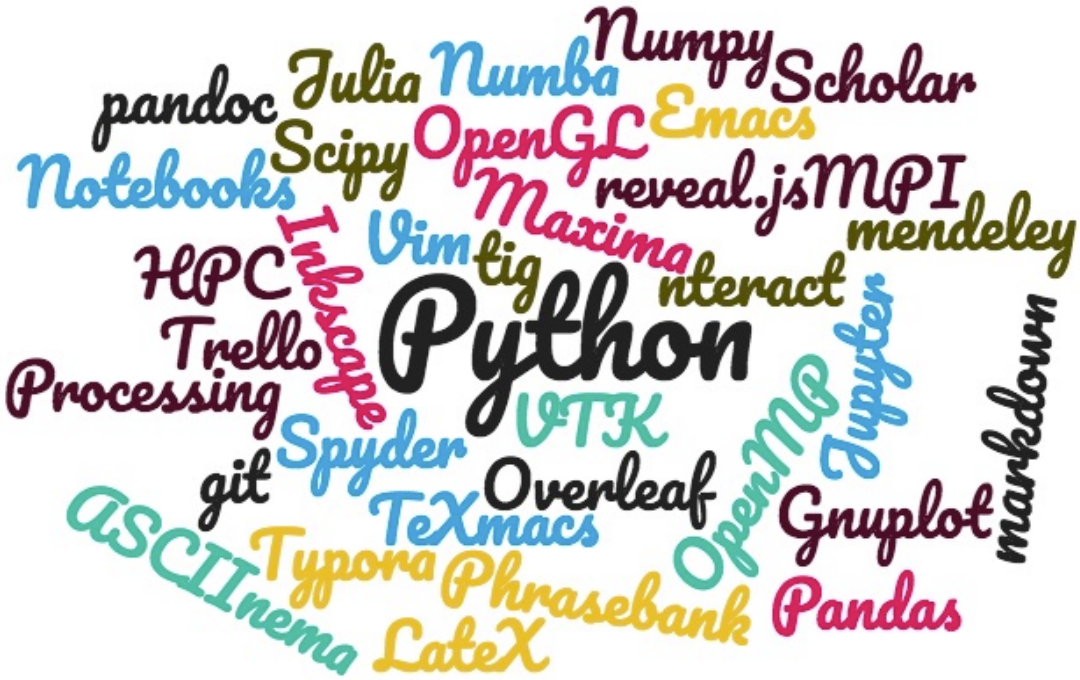
The efficient toolbox of the Computational Scientist



Dr. Gábor Závodszky - *Computational Science Lab*

https://github.com/gzavo/CS_Assignment

> "Never do a live demo" -- Every presenter ever

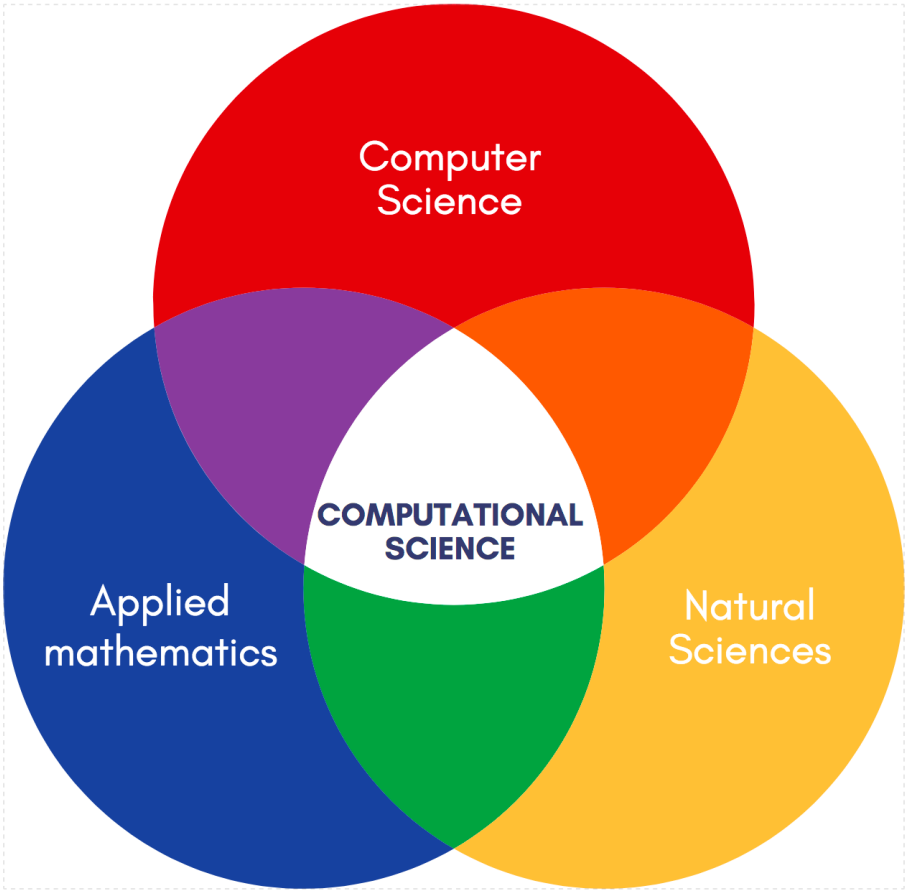
Structure of the lecture

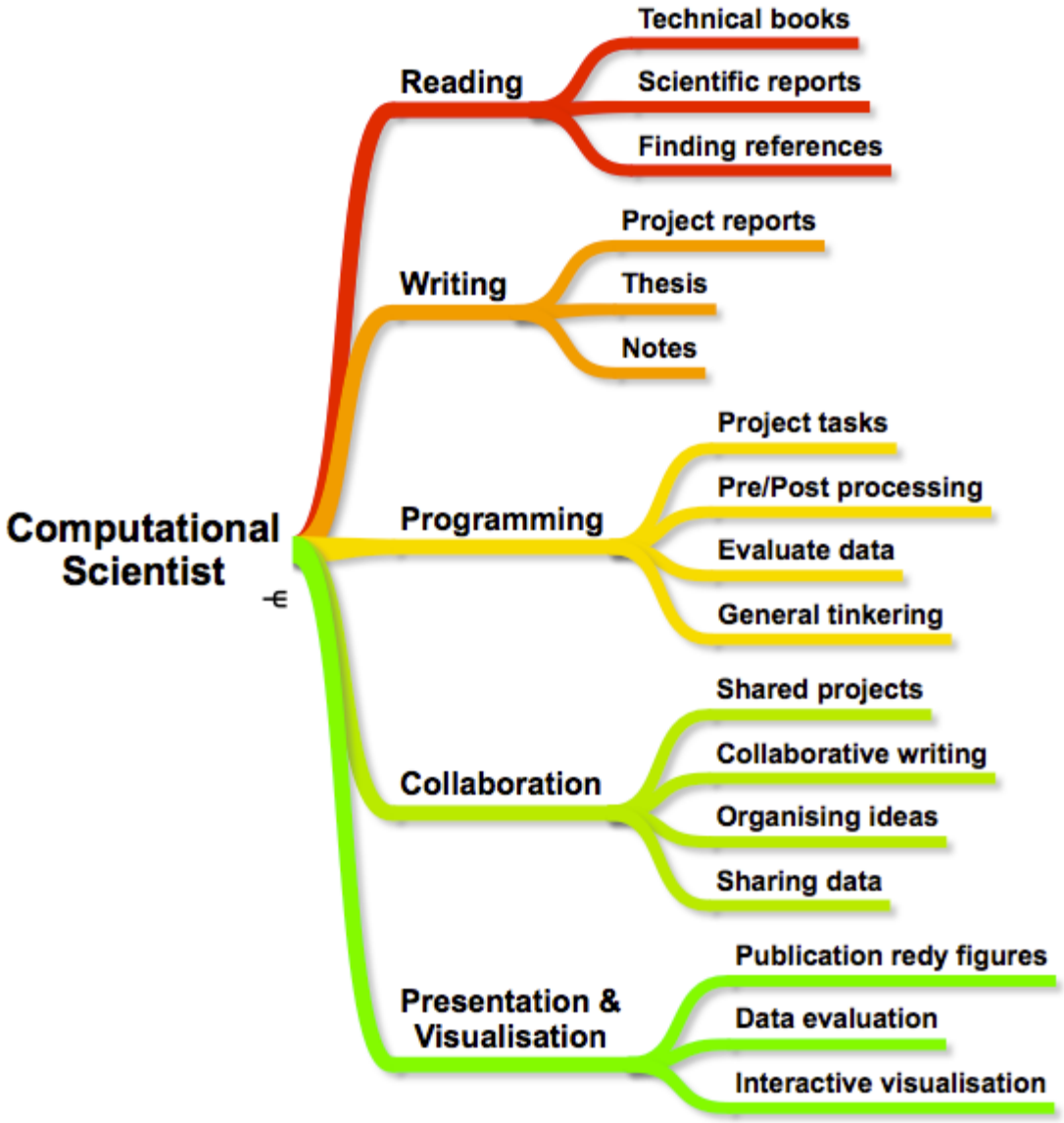
- Domains of "The Computational Scientist"
- Scientific writing and reading
- Programming and visualization basics
- Software development in a collaborative environment
- Homework assingment

- Can be an information overload!
- These slides contain the very basics.
- Due to the structure of this MSc programme, you might not need *everything*.

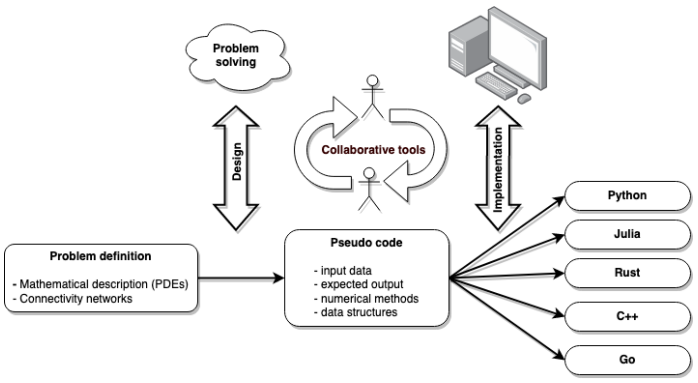
- The most important things are marked: **(!)**.
- Some things are there to hear at least once.
- You can use the pdf version as a list of useful tools.

Who is the Computational Scientist? - Domains and fundamental skills





General workflow of the Computational Scientist



CLS Activities - Reading

- UvA Library (!) (http://uba.uva.nl/en)
- You can also use it to get access to non-open access journal papers.

- Google Scholar (!) - https://scholar.google.com
  - Search for papers
  - export references
  - look at researcher profiles
  - E.g.: "Carbon monoxide, boldly goes where..."
- Web of Science, Scopus, ...

- Managing references: Mendeley, **Zotero** (!), Papers, etc.
  - Sync. paper database
  - Annotate pdfs, add notes and sync them as well
- Most students use Zotero (I use it as well).
- Mendeley gives more cloud space, but its less flexible.

CLS Activities - Writing

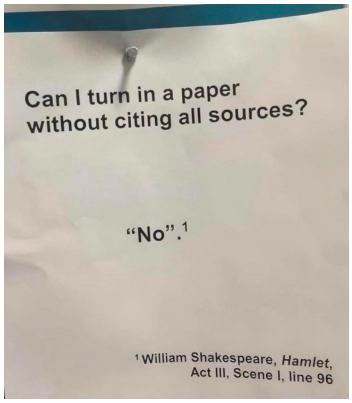
- LaTeX
  - Overleaf (!) - https://www.overleaf.com/
  - TexMaker
  - TeXmacs
  - latex2png - http://latex2png.com/
- Typist (modern LaTeX alternative written in Rust) https://typst.app/

-> Why is LaTeX useful?

- Not a WYSIWYG solution
- But often WYGIWYN
- Abundance of templates
- Literally a programming language for word processing

In [ ]:

Remember...



from IPython.display import IFrame

IFrame("https://www.nature.com/news/scientific-writing-the-online-cooperative-1.16039", "100%", 600)

- Markdown (!)
  - Use Pandoc (!) to turn it to .doc, .pdf, .html, you name it (https://pandoc.org/)
  - Zettlr (!) (https://www.zettlr.com/)
  - Typora (https://typora.io/)
  - Mark Text (https://github.com/marktext/marktext)

- Basically everywhere (websites, forums, editors...really, everywhere)
- Other supersets / alternatives (ASCIIDOC, ...)

Markdown

Write equations: $e^{i\pi} + 1 = 0$

Embed code:

```
def f(x):
    """docstring of this very useful function"""
    return x**2
```

Add tables:

This	is
a	table

Create lists:

- list item 1
- list item 2
- list item 3

Markdown is a simple way to format text that looks great on any device. It doesn't do anything fancy like change the font size, color, or type — just the essentials, using keyboard symbols you already know.

TRY OUR 10 MINUTE MARKDOWN TUTORIAL

Type	Or	... to Get
`*Italic*`	`_Italic_`	*Italic*
`**Bold**`	`__Bold__`	**Bold**
`# Heading 1`	`Heading 1` `==========`	Heading 1
`## Heading 2`	`Heading 2` `---------`	Heading 2

`[Link](http://a.com)`	`[Link][1]`
	⋮
	`[1]: http://b.org`
		Link

`![Image](http://url/a.png)`	`![Image][1]`
	⋮
	`[1]: http://url/b.jpg`
		M↓

`> Blockquote`		Blockquote
`* List` `* List` `* List`	`- List` `- List` `- List`	• List • List • List
`1. One` `2. Two`	`1) One` `2) Two`	1. One 2. Two

from IPython.display import IFrame IFrame("http://commonmark.org/help/", "100%", 600)

CLS Activities - Writing - cont.

Pandoc a universal document converter

Donate Flattr

- About
- Installing
- Getting started
- Demos
- Documentation
- Help
- Extras
- Releases

About pandoc

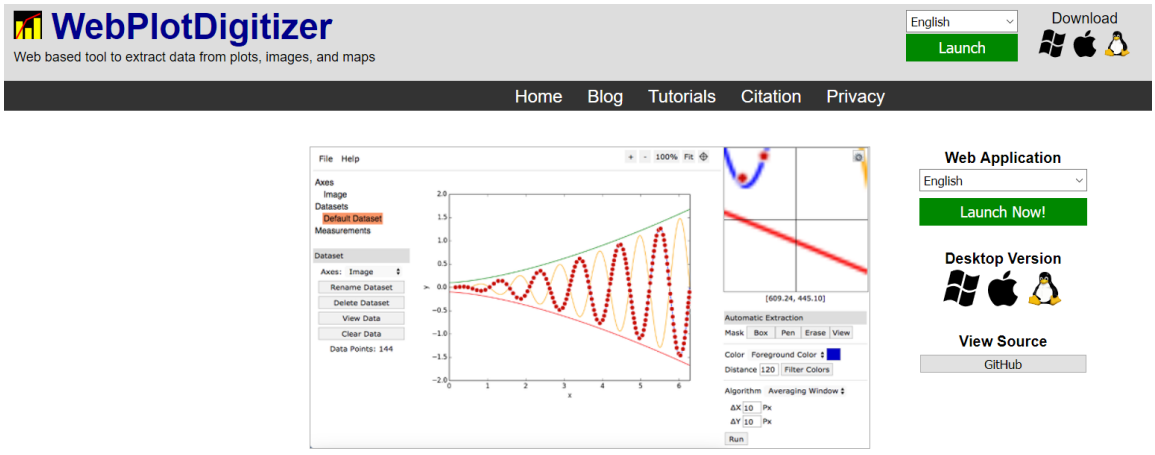
If you need to convert files from one markup format into another, pandoc is your swiss-army knife. Pandoc can convert documents in (several dialects of) Markdown, reStructuredText, textile, HTML, DocBook, LaTeX, MediaWiki markup, TWiki markup, TikiWiki markup, Creole 1.0, Vimwiki markup, OPML, Emacs Org-Mode, Emacs Muse, txt2tags, Microsoft Word docx, LibreOffice ODT, EPUB, or Haddock markup to

HTML formats
XHTML, HTML5, and HTML slide shows using Slidy, reveal.js, Slideous, S5, or DZSlides

Word processor formats
Microsoft Word docx, OpenOffice/LibreOffice ODT, OpenDocument XML, Microsoft PowerPoint.

Ebooks
EPUB version 2 or 3, FictionBook2

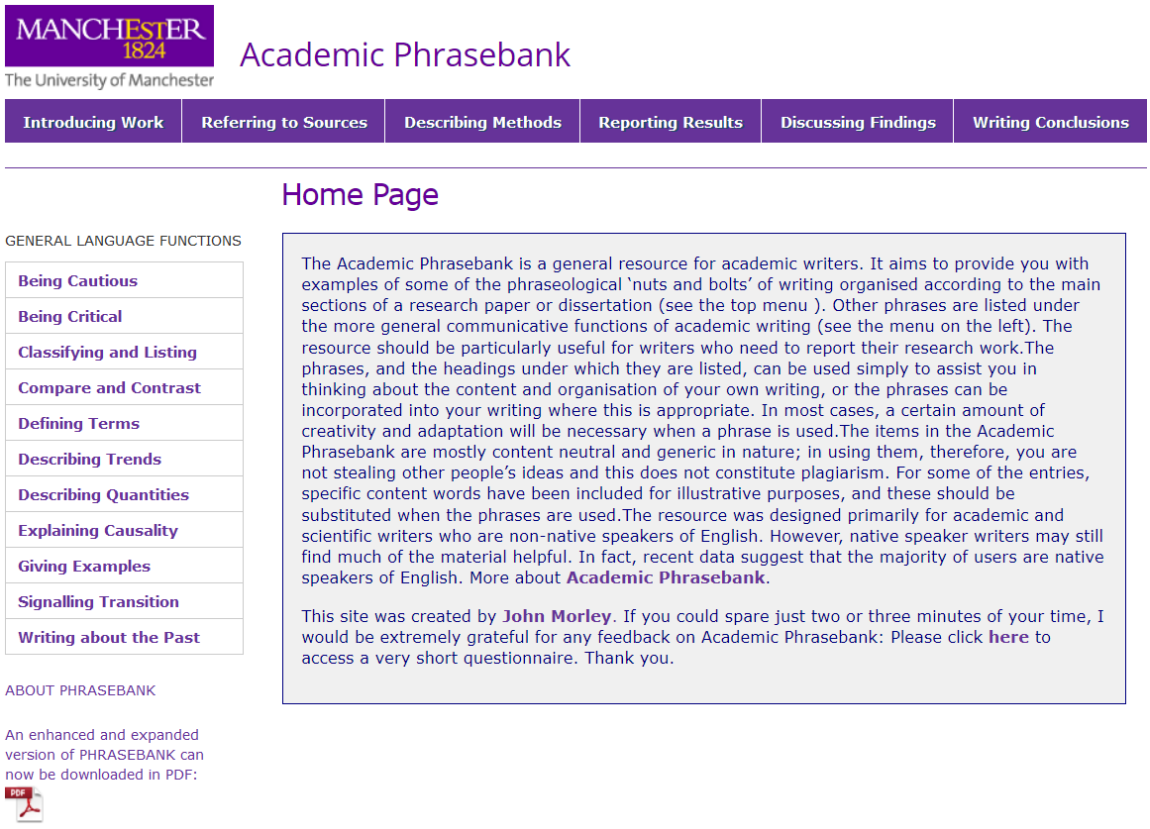
https://automeris.io/WebPlotDigitizer/



It is often necessary to reverse engineer images of data visualizations to extract the underlying numerical data. WebPlotDigitizer is a semi-automated tool that makes this process extremely easy:

- Works with a wide variety of charts (XY, bar, polar, ternary, maps etc.)
- Automatic extraction algorithms make it easy to extract a large number of data points
- Free to use, opensource and cross-platform (web and desktop)
- Used in hundreds of published works by thousands of users
- Also useful for measuring distances or angles between various features
- More to come soon...

http://www.phrasebank.manchester.ac.uk/



CLS Activities - Programming

- programming languages

- editors
- examples

Programming languages

- Python (!)
- C (!) (performance, hardware access)
- C++ (!) (If you need performance + OO)
- Julia (!) (C + Python + Lisp structures)
- Rust (!) (performance and safety)
- Kotlin (primarily mobile development)
- Nim (Python, but C)
- R (statistics)
- Scala (data science)
- Go (Google's try)
- Javascript (too popular to leave out)
- Racket (for the gourmet)
- +1 Lobster lang. (http://strlen.com/lobster/)

> Application domains can overlap, choose 2-3 and learn them well!

> For instance, C/C++ and Python is a quite versatile combination.

> Computer Language Benchmark Game (https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html)

> Rosetta Code (http://www.rosettacode.org)

In [ ]:

General editors

- **Visual Studio Code** (!!)
- Sublime Text
- **Jupyter** / **JupyterLab** (!)
- Cursor, Roo, ... zoo of AI assisted IDEs
- Vi/Vim/Neo Vim
- Emacs (Spacemacs)
- micro
- Note++
- ... wide palett of other editors ...

Specific editors / IDEs

- Spyder (!) - https://www.spyder-ide.org/
- **Pluto** (for Julia) (!) - https://plutojl.org/
- PyCharm
- Juno (VSCode, Julia) - https://www.julia-vscode.org/
- Lazarus (UI for desktop) - https://www.lazarus-ide.org/
- Code::Blocks - https://www.codeblocks.org/

In practice

Why is Jupyter important?

- HTML5 platform (kernel as backend, can run remotely on a different machine)
- Compatible with several languages (C++, Python, Julia, Haskell, ...)
- Important step in hosting and sharing codes (reproducability)
- Towards reproducable science! (also see Jupyter Lab)

> Careful: non-linear state, see next example! (Check out Pluto for linear state solution)

- Additional benefits of the separate backend: parallel execution, cluster management, remote execution

In [4]:
```
b=2
print(b)
```

```
2
```

In [5]:
```
print(b)
b=3
```

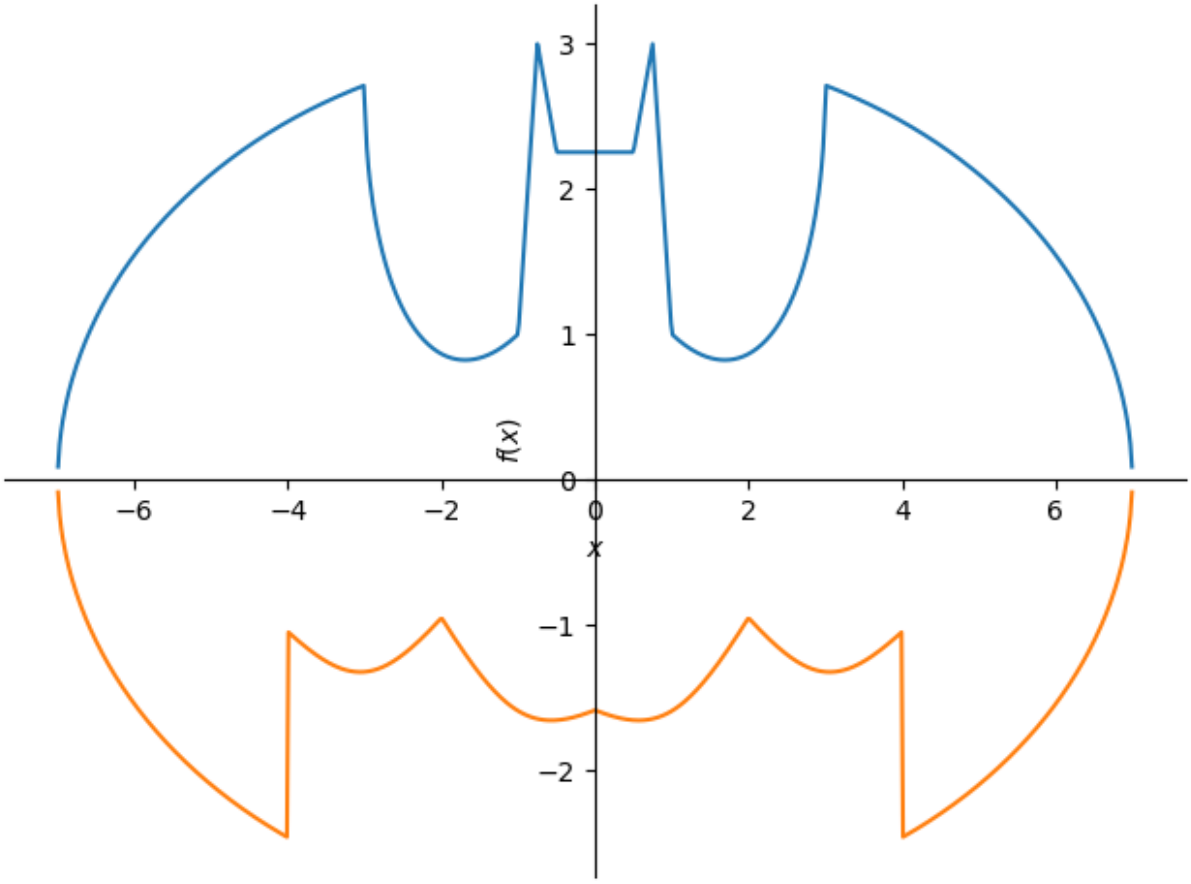
```
2
```

In [1]:
```
import sympy as sp
sp.init_printing(use_latex='mathjax')
x,y,z = sp.symbols('x,y,z')
f = sp.sin(x*y)+sp.cos(y*z)
sp.integrate(f,x)
```

Out[1]:

$$x\cos\left(yz\right) + \begin{cases} -\frac{\cos\left(xy\right)}{y} & \text{for } y \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

In [2]:
```
import sympy as sm
from sympy.plotting import plot
x = sm.symbols('x', real=True)
h_ = sm.symbols('h_')

w = 3 * sm.sqrt(1 - (x / 7)**2)
l = ((x + 3) / 2 - sm.S(3) / 7 * sm.sqrt(10) * sm.sqrt(4 - (x + 1)**2) +
    sm.S(6) / 7 * sm.sqrt(10))
r = ((3 - x) / 2 - sm.S(3) / 7 * sm.sqrt(10) * sm.sqrt(4 - (x - 1)**2) +
    sm.S(6) / 7 * sm.sqrt(10))
h = (3*(sm.Abs(x - sm.S.Half) + sm.Abs(x + sm.S.Half) + 6) -
             11*(sm.Abs(x - sm.S(3)/4) + sm.Abs(x + sm.S(3)/4)))/2
f = ((h - l) * sm.Heaviside(x + 1, 0) +
    (r - h) * sm.Heaviside(x - 1, 0) +
    (l - w) * sm.Heaviside(x + 3, 0) +
    (w - r) * sm.Heaviside(x - 3, 0) +
    w)
g = (sm.S(1) / 2 * (sm.Abs(x / 2) + sm.sqrt(1 + (sm.Abs(sm.Abs(x) - 2) -
    1)**2) - sm.S(1) / 112 * (3 * sm.sqrt(33) - 7) * x**2 + 3 *
    sm.sqrt(1 - (sm.S(1)/ 7 * x)**2) - 3) * ((x + 4) / sm.Abs(x + 4) -
    (x - 4) / sm.Abs(x - 4)) - 3 * sm.sqrt(1 - (x / 7)**2))
sm.plot(f, g)
```



Out[2]: <sympy.plotting.backends.matplotlibbackend.matplotlib.MatplotlibBackend at 0x105ed1850>

SymPy Cheatsheet (http://sympy.org)

Basics

Sympy help:	`help(function)`
Declare symbol:	`x = Symbol('x')`
Substitution:	`expr.subs(old, new)`
Numerical evaluation:	`expr.evalf()`
Expanding:	`expr.expand()`
Common denominator:	`ratsimp(expr)`
Simplify expression:	`simplify(expr)`

Constants

π:	`pi`
e:	`E`
∞:	`oo`
i:	`I`

Numbers types

Integers ($\mathbb{Z}$):	`Integer(x)`
Rationals ($\mathbb{Q}$):	`Rational(p, q)`
Reals ($\mathbb{R}$):	`Float(x)`

Basic funtions

Trigonometric:	`sin cos tan cot`
Cyclometric:	`asin acos atan acot`
Hyperbolic:	`sinh cosh tanh coth`
Area hyperbolic:	`asinh acosh atanh acoth`
Exponential:	`exp(x)`
Square root:	`sqrt(x)`
Logarithm ($\log_b a$):	`log(a, b)`
Natural logarithm:	`log(a)`
Gamma ($\Gamma(x)$):	`gamma(x)`
Absolute value:	`abs(x)`

Calculus

$\lim_{x\to a} f(x)$:	`limit(f, x, a)`
$\lim_{x\to a^-} f(x)$:	`limit(f, x, a, dir='-')`
$\lim_{x\to a^+} f(x)$:	`limit(f, x, a, dir='+')`
$\frac{d}{dx} f(x)$:	`diff(f, x)`
$\frac{\partial}{\partial x} f(x, y)$:	`diff(f, x)`
$\int f(x)\,dx$:	`integrate(f, x)`
$\int_a^b f(x)\,dx$:	`integrate(f, (x, a, b))`
Taylor series (at a, deg n)	`f.series(x, a, n)`

Equations

Equation $f(x) = 0$:	`solve(f, x)`
System of equations:	`solve([f, g], [x, y])`
Differential equation:	`dsolve(equation, f(x))`

Geometry

Points:	`a = Point(xcoord, ycoord)`
Lines:	`l = Line(pointA, pointB)`
Circles:	`c = Circle(center, radius)`
Triangles:	`t = Triangle(a, b, c)`
Area:	`object.area`
Intersection:	`intersection(a, b)`
Checking tangency:	`c.is_tangent(l)`

Plotting

Plot:	`Plot(f, [a, b])`
Zoom: +/−:	`R/F or PgUp/PgDn or Numpad +/-`
Rotate X,Y axis:	`Arrow Keys or WASD`
Rotate Z axis:	`Q and E or Numpad 7 and 9`
View XY:	F1
View XZ:	F2
View YZ:	F3
View Perspective:	F4
Axes Visibility:	F5
Axes Colors:	F6
Screenshot:	F8
Exit plot:	ESC

Discrete math

Factorial ($n!$):	`factorial(n)`
Binomial coefficient $\binom{n}{k}$:	`binomial(n, k)`
Sum ($\sum_{n=a}^{b} expr$):	`summation(expr, (n, a, b))`
Product ($\prod_{n=a}^{b} expr$):	`product(expr, (n, a, b))`

Linear algebra

Matrix definition:	`m = Matrix([[a, b], [c, d]])`
Determinant:	`m.det()`
Inverse:	`m.inv()`
Identity matrix $n \times n$:	`eye(n)`
Zero matrix $n \times n$:	`zeros(n)`
Ones matrix $n \times n$:	`ones(n)`

Printing

LaTeX print:	`print latex()`
Python print:	`print python()`
Pretty print:	`pprint()`

Examples

Find 100 digits of π^e:
`(pi**E).n(100)`

Expand $(x+y)^2(x-y)(x^2+y)$:
`((x + y)**2 * (x - y) * (x**2 + y)).expand()`

Simplify $\frac{1}{x} + \frac{x \sin x - 1}{x^2 - 1}$:
`simplify((1/x) + (x * sin(x) - 1)/(x**2 - 1))`

Check if line passing through points $(0, 1)$ and $(1, 1)$ is tangent to circle with center at $(5, 5)$ and radius 3:
`Circle(Point(5,5), 3).is_tangent(`
`Line(Point(0,1), Point(1,1)))`

Find roots of $x^4 - 4x^3 + 2x^2 - x = 0$:
`solve(x**4 - 4*x**3 + 2*x**2 - x, x)`

Solve the equations system: $x + y = 4$, $xy = 3$:
`solve([x + y - 4, x*y - 3], [x, y])`

Calculate limit of the sequence $\sqrt[n]{n}$:
`limit(n**(1/n), n, oo)`

Calculate left-sided limit of the function $\frac{|x|}{x}$ in 0:
`limit(abs(x)/x, x, 0, dir='-')`

Calculate the sum $\sum_{n=0}^{100} n^2$:
`summation(n**2, (n, 0, 100))`

Calculate the sum $\sum_{n=0}^{\infty} \frac{1}{n^2}$:
`summation(1/n**2, (n, 0, oo))`

Calculate the integral $\int \cos^3 x\,dx$:
`integrate(cos(x)**3, x)`

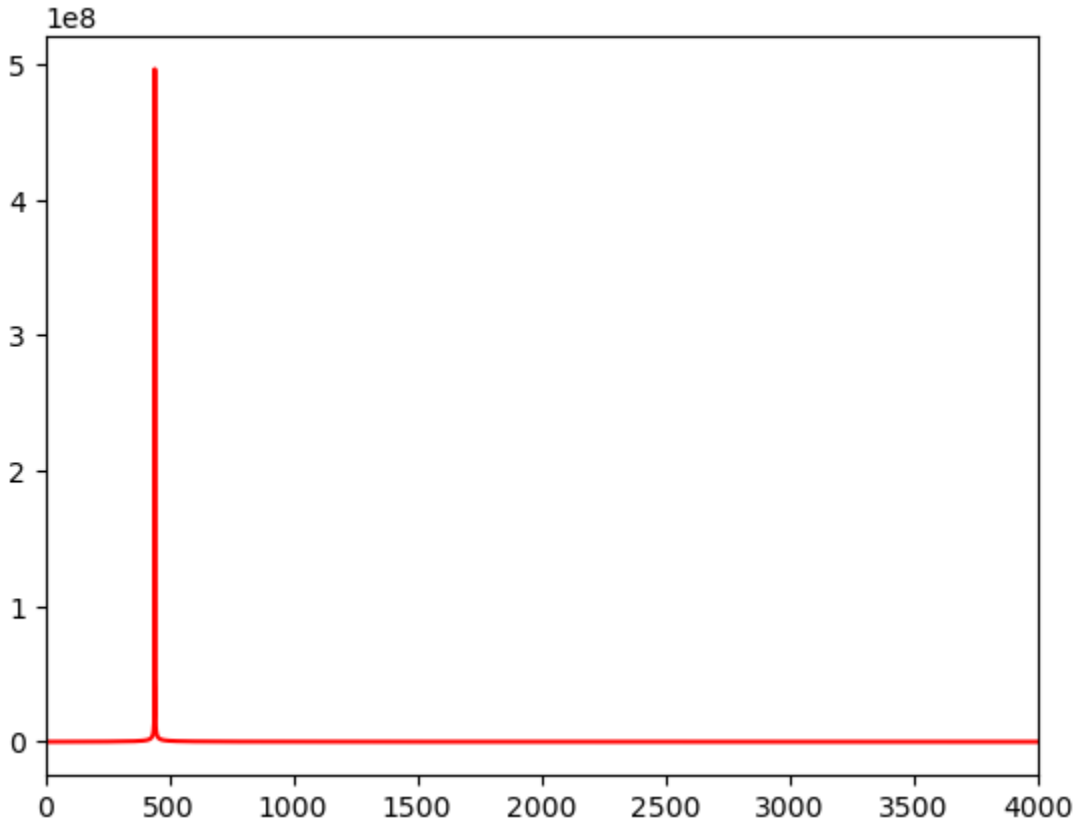
Calculate the integral $\int_1^{\infty} \frac{dx}{x^2}$:
`integrate(1/x**2, (x, 1, oo))`

Find 10 terms of series expansion of $\frac{1}{1-2x}$ at 0:
`(1/(1 - 2*x)).series(x, 0, 10)`

Solve the differential equation $f''(x) + 9f(x) = 1$:
`dsolve(f(x).diff(x, x) + 9*f(x) - 1, f(x))`

In [3]:
```
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.fftpack import fft; from scipy.io import wavfile
fs, data = wavfile.read('sound/sine440.wav') # load the data, 16 bit, 44.1 k
c = fft(data.T) # calculate fourier transform (complex numbers list)
d = len(c)//2  # you only need half of the fft list (real signal symmetry)
plt.plot(abs(c[:(d-1)]),'r'); plt.xlim((0,4000))
```

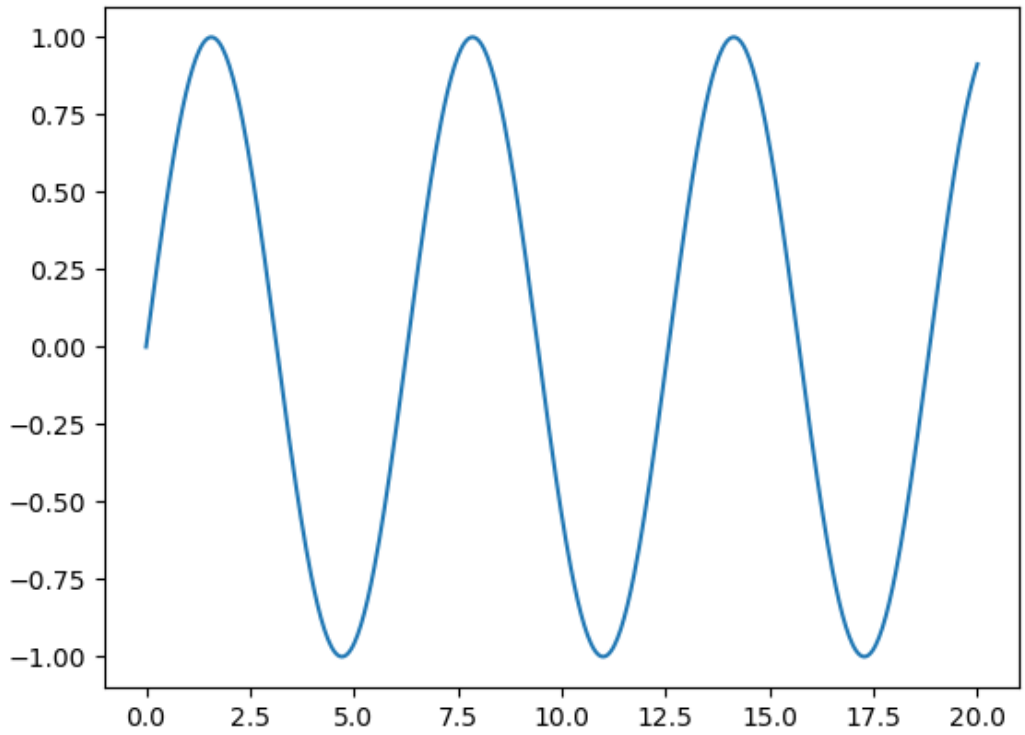
Out[3]: (0.0, 4000.0)



In [4]:
```
import IPython
IPython.display.Audio('sound/sine440.wav')
# IPython.display.Audio('sound/sample.wav')
```

Out[4]: 0:00 / 0:01

In [5]:
```
import matplotlib.pyplot as plt
%matplotlib notebook
import numpy as np
x = np.linspace(0,20,500)
plt.plot(x, np.sin(x))
```



Out[5]: [<matplotlib.lines.Line2D at 0x1347d4080>]

In [6]:
```
import matplotlib.pyplot as plt
%matplotlib inline
from ipywidgets import interact, IntSlider
from IPython.display import display,clear_output

def f(freq):
    x = np.linspace(0,20,500)
    plt.plot(x, np.sin(x*freq))
    plt.show()
    #display(plt.figure)

interact(f, freq=IntSlider(min=1,max=5,step=1,value=1));
```

interactive(children=(IntSlider(value=1, description='freq', max=5, min=1), Output()), _dom_classes=('widget-i…

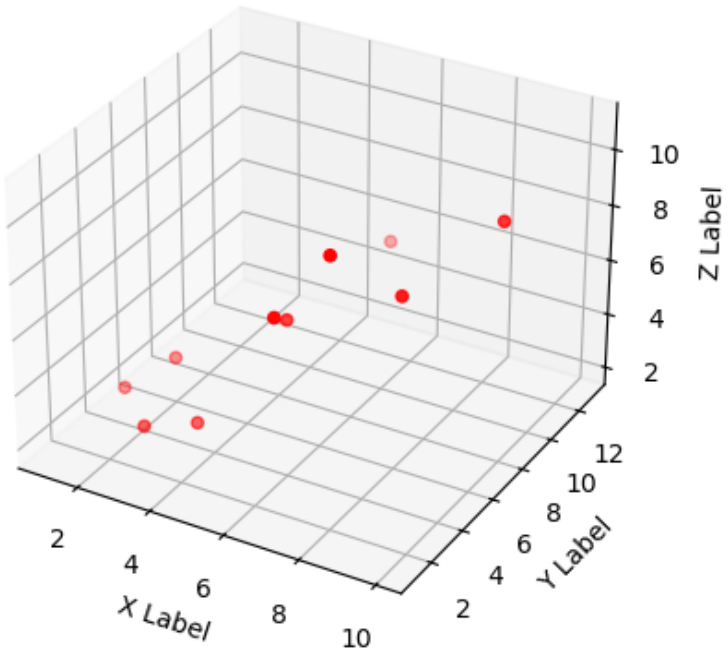
In [7]:
```
from IPython.display import display, Javascript
import ipywidgets as widgets

L = widgets.Label("Hello World")
display(L)
```

Label(value='Hello World')

In [17]:
```
L.value = "Howdy World"
```

In [8]:
```
from mpl_toolkits.mplot3d import Axes3D; import matplotlib.pyplot as plt;
%matplotlib notebook
```

```
fig = plt.figure(); ax = fig.add_subplot(111, projection='3d')
x =[1,2,3,4,5,6,7,8,9,10]; y =[5,6,2,3,13,4,1,2,4,8]; z =[2,3,3,3,5,7,9,11,9
ax.scatter(x, y, z, c='r', marker='o'); ax.set_xlabel('X Label'); ax.set_yla
```



Out[8]: Text(0.5, 0, 'Z Label')

In [9]:
```
import ipyvolume as ipv
import numpy as np
import ipyvolume.datasets
stream = ipyvolume.datasets.animated_stream.fetch()
fig = ipv.figure()
# instead of doing x=stream.data[0], y=stream.data[1], ... vz=stream.data[5]
# limit to 50 timesteps to avoid having a huge notebook
q = ipv.quiver(*stream.data[:,0:50,:200], color="red", size=7)
ipv.style.use("dark") # looks better
ipv.animation_control(q, interval=200)
ipv.show()
```

```
Container(children=[HBox(children=(Play(value=0, interval=200, max=49), IntS
lider(value=0, max=49)))], figure=…
```

In [10]:
```
import matplotlib.pyplot as plt
%matplotlib notebook
import pandas as pd
import seaborn as sns
df = sns.load_dataset("anscombe") #Anscombe's quartet, there are others (Tit
df.head()
```

Out[10]:

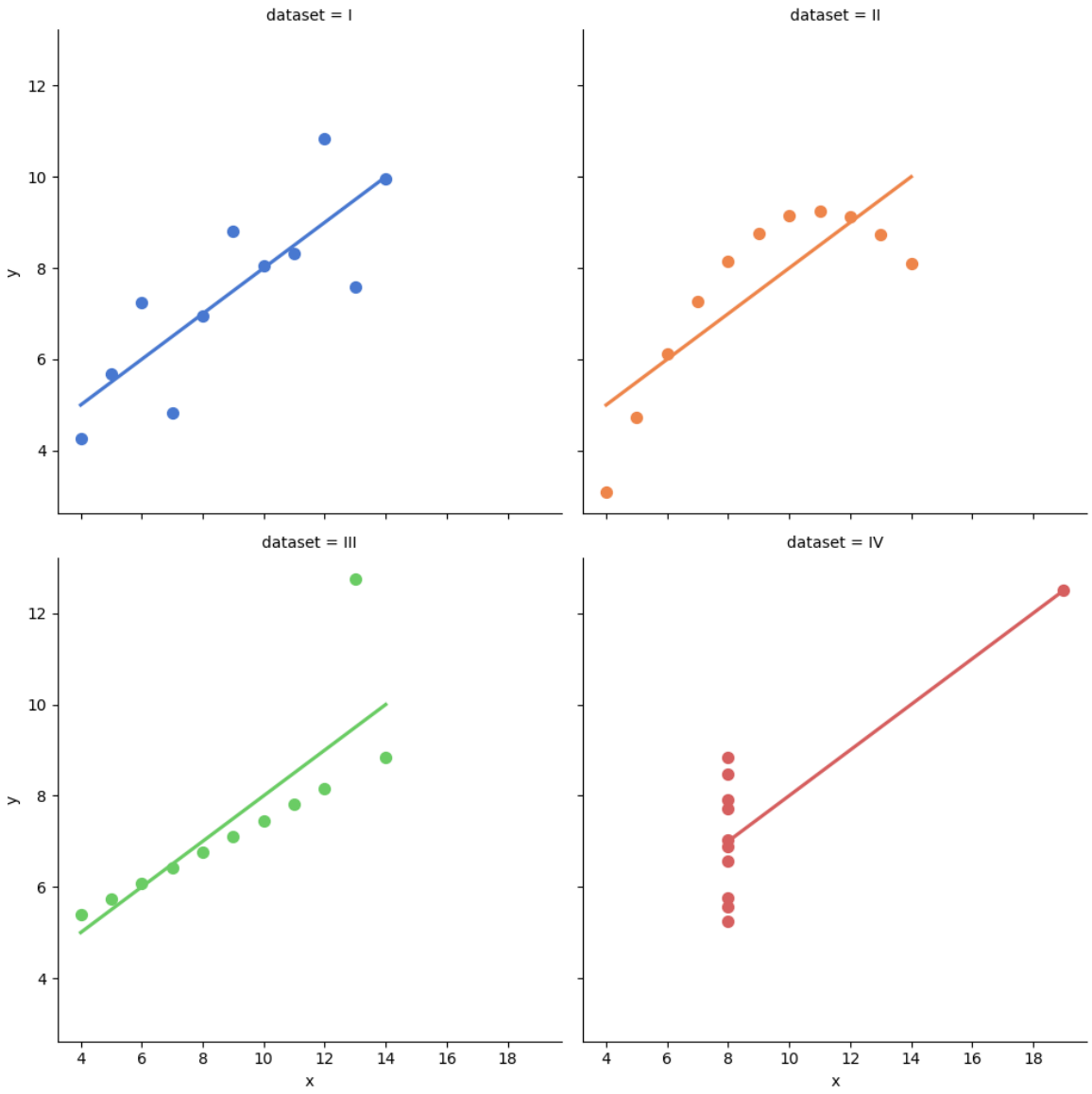
	dataset	x	y
0	I	10.0	8.04
1	I	8.0	6.95
2	I	13.0	7.58
3	I	9.0	8.81
4	I	11.0	8.33

In [11]:
```
df.groupby(df.dataset).describe()
```

Out[11]:

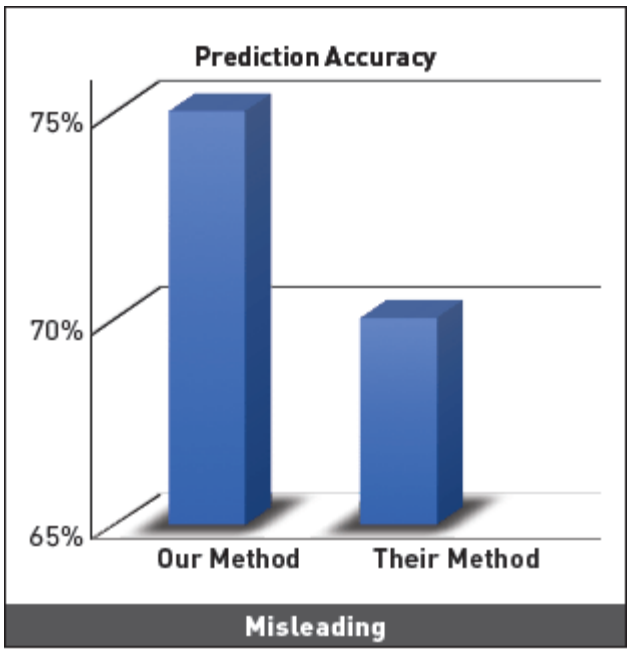
	x										
	count	mean	std	min	25%	50%	75%	max	count	mean	
dataset											
I	11.0	9.0	3.316625	4.0	6.5	9.0	11.5	14.0	11.0	7.500909	2.031
II	11.0	9.0	3.316625	4.0	6.5	9.0	11.5	14.0	11.0	7.500909	2.031
III	11.0	9.0	3.316625	4.0	6.5	9.0	11.5	14.0	11.0	7.500000	2.030
IV	11.0	9.0	3.316625	8.0	8.0	8.0	8.0	19.0	11.0	7.500909	2.030

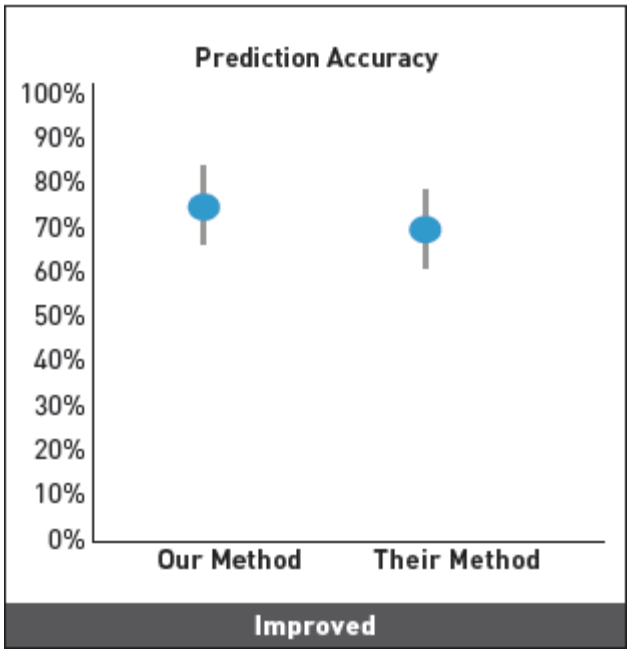
In [12]:
```
%matplotlib notebook
# To put any warning below
sns.lmplot(x="x", y="y", col="dataset", hue="dataset", data=df, col_wrap=2,
           scatter_kws={"s": 50, "alpha": 1}); plt.show()
```

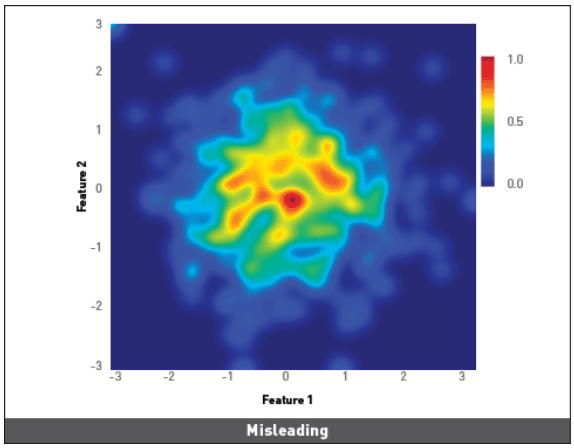


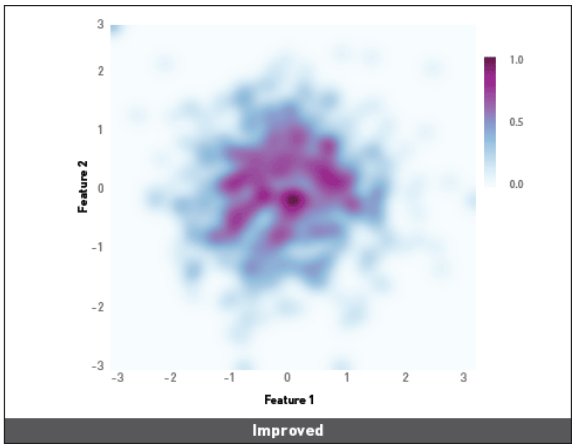
Visualization practices

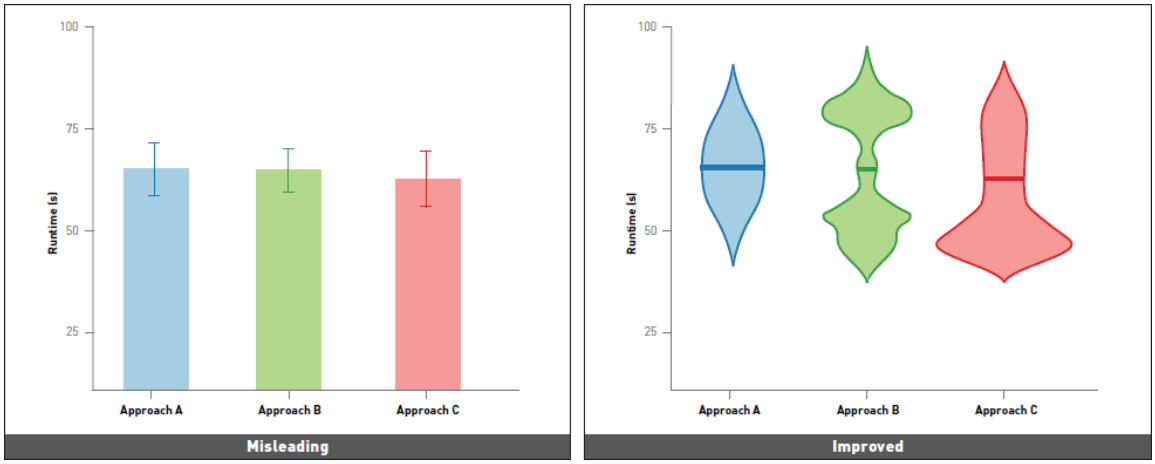
https://interactions.acm.org/archive/view/july-august-2018/the-good-the-bad-and-the-biased


Prediction Accuracy
75%
70%
65%
Our Method
Their Method
Misleading


Prediction Accuracy
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Our Method
Their Method
Improved


Feature 2
Feature 1
Misleading


Feature 2
Feature 1
Improved



Further visualization tools

Because our simulations produce a lot of data (big data?!), but the purpose of the computation is not numbers, but insight.

- python matplotlib (!) - (https://matplotlib.org/)
- ParaView (!) - VTK based parallel 3D visualization tool (https://www.paraview.org/)
- MayaVi - ParaView alternative, written in python (embeddable!) (https://docs.enthought.com/mayavi/mayavi/)
- PyVista - Practically the new VTK binding (https://docs.pyvista.org/)
- gnuplot (!) - Swiss army knif of plotters (http://www.gnuplot.info/)
- vedo - Simple visualization 2D/3D, simulation friendly (https://github.com/marcomusy/vedo)
- Processing - Programming language designed for 3D visualizations (https://processing.org/)
- Inkscape - For that nice poster. (https://inkscape.org/)
- Blender - (https://www.blender.org/)

Presentation tools

- MS PowerPoint (!)
- LaTex (!) (e.g. Beamer)
- Reveal.js (https://github.com/hakimel/reveal.js/)
- Jupyter Notebook (sort of works...)
- TexMacs, LyX, ...

Further things to look at in the Python world

- https://github.com/barbagroup/CFDPython

- http://mbakker7.github.io/exploratory_computing_with_python
- https://github.com/vinta/awesome-python

Other tools to mention

- Desktop Jupyter: nteract (https://nteract.io/)
- Desktop JupyterLab: (https://github.com/jupyterlab/jupyterlab-desktop)
- Calculator: SpeedCrunch (https://speedcrunch.org/)
- CAS: Sympy (!) (www.sympy.org), Maxima (http://maxima.sourceforge.net/)
- Worksheet: SMathStudio (https://en.smath.com/view/SMathStudio/summary)
- Recording terminal session: ASCIInema (https://asciinema.org/)
- Collection of command line terminal tools (https://www.wezm.net/technical/2019/10/useful-command-line-tools/)

Software development in a collaborative environment

- Student-ware ('agile' method? - https://www.atlassian.com/agile)
- Guidelines PEP8
- Version control
- Tests

Software development as a student

The boundary conditions are a bit different compared to 'real' development, but still aim to adopt good practices!

The typical situation:

- most often no preliminary design
- "let's see what happens" first version
- then the code is "grown" in incremental steps
- "backups" is some old version on a pendrive, or sent in email
- patchworky design in team development
- this is a natural process given the boundary conditions

Pro.:

- nothing really
- maybe time efficiency on the short-term (*maybe*)

Con.:

- difficult collaboration with unnecessary friction
- quickly leads to decaying efficiency
- resulting code is not robust or future proof
- often difficult to extend (aka. rewrite to add a feature)

What can be improved?

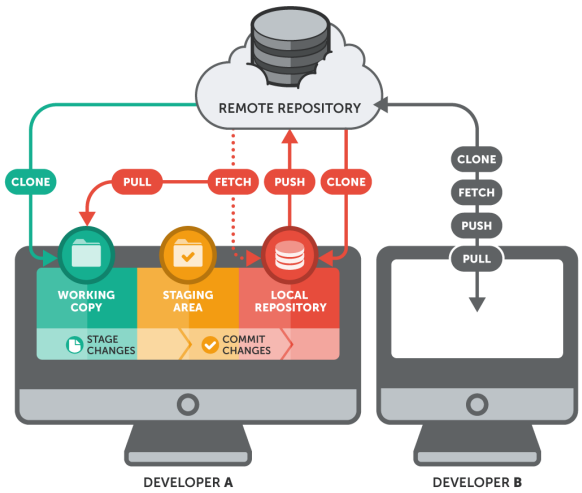
Coding style (!)

- Style guides:
  - Python PEP8 (https://www.python.org/dev/peps/)
  - C++ Google Style guide (https://google.github.io/styleguide/cppguide.html)
  - C - Many, e.g. Rob Pike's (https://www.maultech.com/chrislott/resources/cstyle/pikestyle.html)
  - Most IDEs have builtin support for this.
- Documentation (proper README.md)
- Code comments and function documentation (e.g. docstring).

In [23]:
```
print("".join(map(lambda x: chr((lambda p, x: int(sum(map(lambda i: p[i]*x**
```

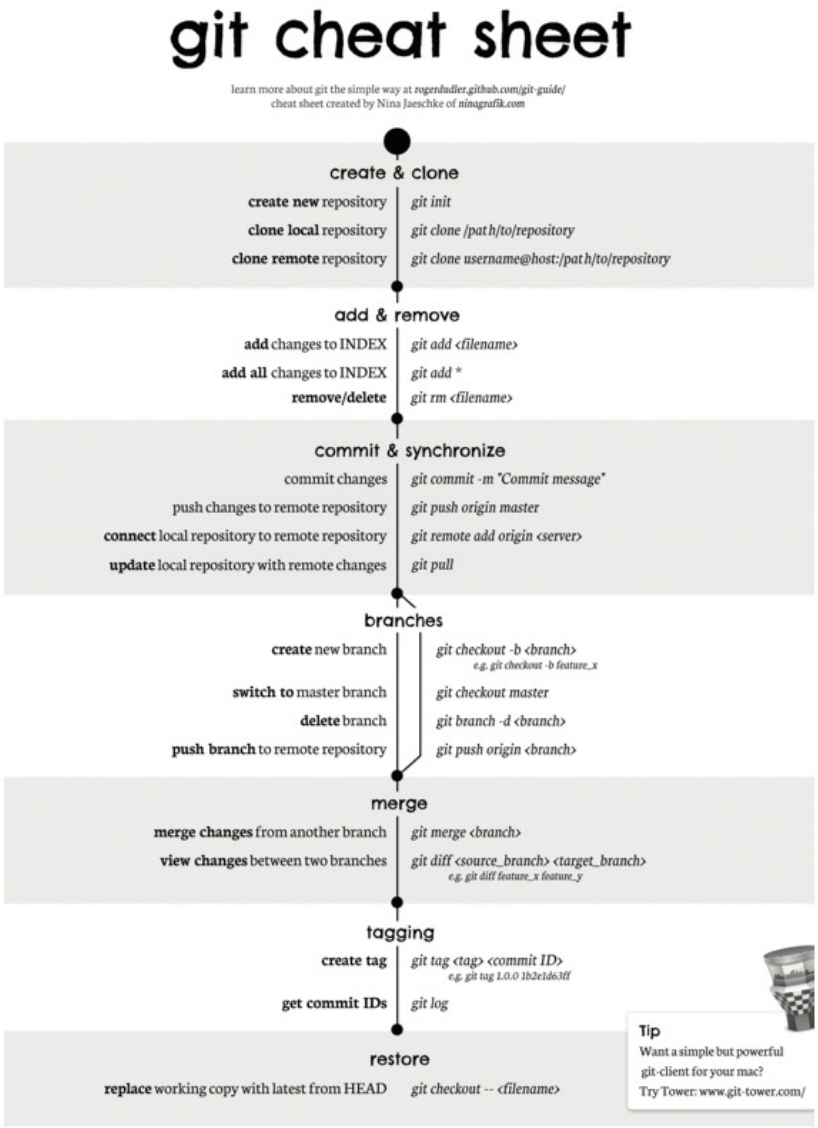
```
Hello world!
```

Version control - git (!)



In [24]:
```
from IPython.display import IFrame
IFrame("doc/git_cheat_sheet.jpg", width=800, height=600) # from http://roger
```

Out[24]:



Tools for git

- tig (!) (https://github.com/jonas/tig)
- sourcetree (https://www.sourcetreeapp.com/)
- GitUp (mac only :/) (https://gitup.co/)
- gitKraken (not free, but part of GitHub Education :/) (https://www.gitkraken.com/)
- most IDEs have built in support for git
- Free git service: github, bitbucket, gitlab

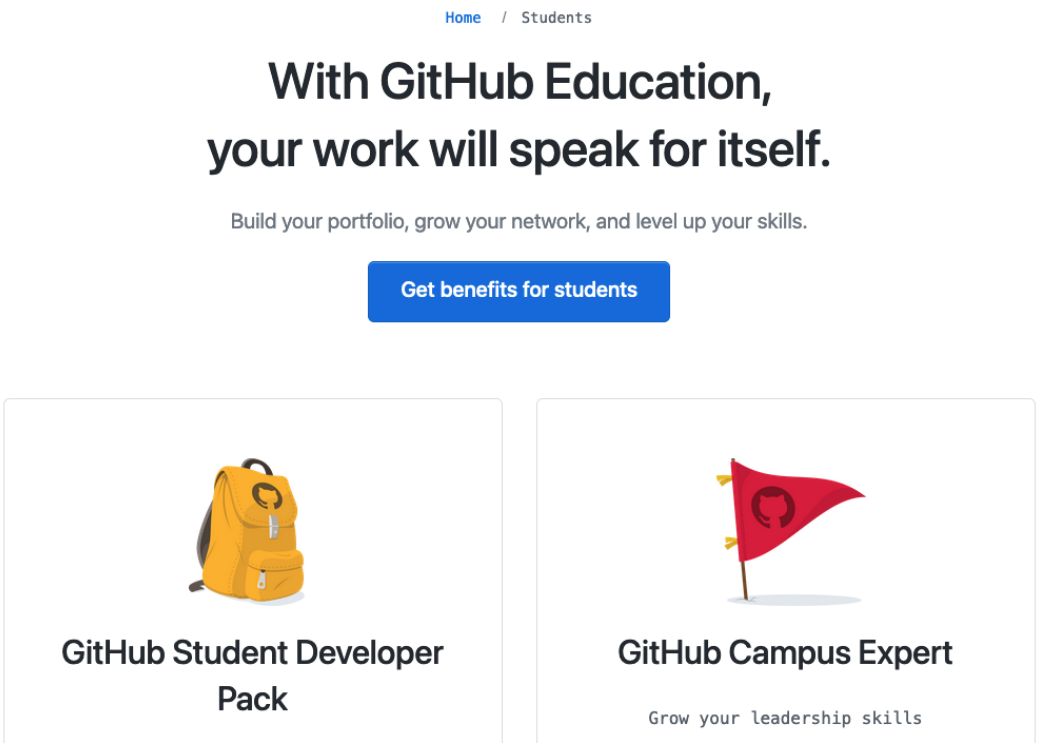
GitHub

https://education.github.com/students

https://docs.github.com/en/get-started/start-your-journey/about-github-and-git

https://git-scm.com/book/en/v2

Access to additional tools: GitHub Copilot, Copilot Chat, Git Kraken, Termius....



Git fame

https://github.com/casperdcl/git-fame

```
~$ git fame --cost hour,month --loc ins
Processing: 100%|████████████████████████| 1/1 [00:00<00:00,  2.16repo/s]
Total commits: 1775
Total ctimes: 2770
Total files: 461
Total hours: 449.7
Total loc: 41659
Total months: 151.0
| Author               |   hrs |   mths |   loc |   coms |   fils | distribution    |
|:---------------------|------:|-------:|------:|-------:|-------:|:----------------|
| Casper da Costa-Luis |   228 |    108 | 28572 |   1314 |    172 | 68.6/74.0/37.3  |
| Stephen Larroque     |    28 |     18 |  5243 |    203 |     25 | 12.6/11.4/ 5.4  |
| pgajdos              |     2 |      9 |  2606 |      2 |     18 | 6.3/ 0.1/ 3.9   |
| Martin Zugnoni       |     2 |      5 |  1656 |      3 |      3 | 4.0/ 0.2/ 0.7   |
| Kyle Altendorf       |     7 |      2 |   541 |     31 |      7 | 1.3/ 1.7/ 1.5   |
| Hadrien Mary         |     5 |      1 |   469 |     31 |     17 | 1.1/ 1.7/ 3.7   |
| Richard Sheridan     |     2 |      1 |   437 |     23 |      3 | 1.0/ 1.3/ 0.7   |
| Guangshuo Chen       |     3 |      1 |   321 |     18 |      7 | 0.8/ 1.0/ 1.5   |
| Noam Yorav-Raphael   |     4 |      1 |   229 |     11 |      6 | 0.5/ 0.6/ 1.3   |
| github-actions[bot]  |     2 |      1 |   186 |      1 |     51 | 0.4/ 0.1/11.1   |
...
```

Testing

- Unit tests (!)
- doc tests
- Property-based testing
- CI (continuous integration) / CD (continuous delivery)

In [15]:
```python
import unittest

def fun(x):
    return x + 1

class MyTest(unittest.TestCase):
    def test(self):
        self.assertEqual(fun(4), 4)

# Testing
# Normally: unittest.main()
unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
F/Users/gzavo/miniforge/envs/a_skills_old/lib/python3.12/site-packages/jupyt
er_client/session.py:203: DeprecationWarning: datetime.datetime.utcnow() is
deprecated and scheduled for removal in a future version. Use timezone-aware
objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  return datetime.utcnow().replace(tzinfo=utc)

======================================================================
FAIL: test (__main__.MyTest.test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/var/folders/cr/kxr8zqds7jsb4nr1fnjr3s1r0000gn/T/ipykernel_41896/135
4764667.py", line 8, in test
    self.assertEqual(fun(4), 4)
AssertionError: 5 != 4

----------------------------------------------------------------------
Ran 1 test in 0.002s

FAILED (failures=1)
```

Out[15]: `<unittest.main.TestProgram at 0x16bd26d80>`

In [16]:
```python
def square(x):
    """Return the square of x.

    >>> square(5)
    9
    >>> square(-2)
    4
    """

    return x * x

# Testing
import doctest
doctest.testmod()
```

```
**********************************************************************
File "__main__", line 4, in __main__.square
Failed example:
    square(5)
Expected:
    9
Got:
    25
**********************************************************************
1 items had failures:
   1 of   2 in __main__.square
***Test Failed*** 1 failures.
```

Out[16]: `TestResults(failed=1, attempted=2)`

https://doi.org/10.25080/Majora-342d178e-016

Falsify your Software: validating scientific code with property-based testing

Zac Hatfield-Dodds[‡*]

Abstract—Where traditional example-based tests check software using manually-specified input-output pairs, property-based tests exploit a general description of valid inputs and program behaviour to automatically search for falsifying examples. Given that Python has excellent property-based testing tools, such tests are often *easier* to work with and routinely find serious bugs that all other techniques have missed.

I present four categories of properties relevant to most scientific projects, demonstrate how each found real bugs in Numpy and Astropy, and propose that property-based testing should be adopted more widely across the SciPy ecosystem.

Index Terms—methods, software, validation, property-based testing

Introduction

Much research now depends on software for data collection, analysis, and reporting; including on software produced and maintained by researchers. This has empowered us enormously: it is hard to imagine an analysis that was possible *at all* a generation ago which could not be accomplished quickly by a graduate student today.

Unfortunately, this revolution in the power and sophistication of our software has largely outstripped work on validation. While *sible to non-experts* in Python, and has added multiple features designed for testing scientific programs[3] since 2019.

What is property-based testing?

Where example-based tests check for an exact expected output, property-based tests make *less precise* but *more general* assertions. By giving up hand-specification of the expected output, we gain tests that can be run on a wide range of inputs.

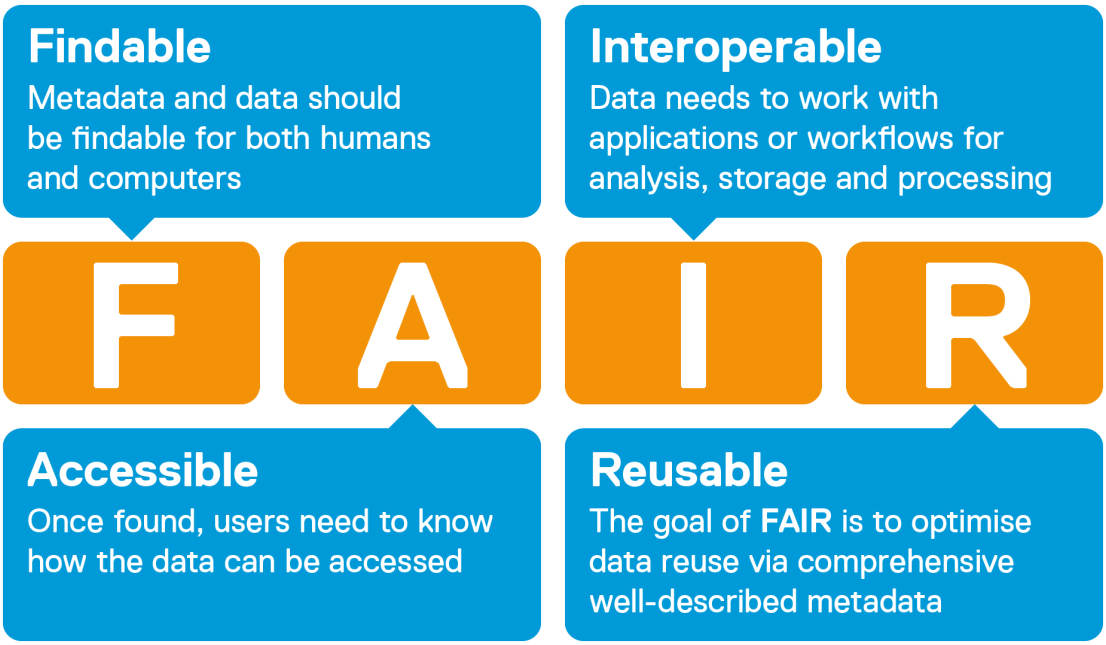
This generality also guides our tests to the right level of abstraction, and gives clear design feedback: where example-based tests map one input to one output regardless of complexity, every special case or interacting feature has to be addressed. Clean abstractions which allow you to say "for all ...", without caveats, stand in clear contrast and are a pleasure to test.

Tests which use random data are usually property-based, but using a library designed for the task has several advantages:

- a concise and expressive interface for describing inputs
- tests are never flaky - failing examples are cached and replayed, even if the test failed on a remote build server
- automatic shrinking facilitates debugging by presenting a minimal failing example for each distinct error

Data handling - FAIR principles

Publication in Nature Data Science - https://doi.org/10.1038/sdata.2016.18



Use case - HemoCell (www.hemocell.eu)

0:00 / 0:04

Some highlights:

HemoCell is a high-cost code, it is in continuous development for 8 years. >20 developers, >20 associated scientific publications, one of the largest distributed execution in EU (330,000 cores).

Tools and techniques of interest:

- Developped in C++ with processing scripts in Python and Bash.
- The source code is version controlled under git (GitHub).
- Uses 2 CI/CD servers to test commits.
- Edited mostly in Visual Studio Code, Sublime Text, and NetBeans.
- The documentation is written in Markdown(-ish) text.
- The publications are written in Overleaf.
- Data visualization through ParaView and Blender.
- Data evaluation through Python (+ HDF5, VTK, ...).
- Presentations, posters done in PowerPoint.
- Illustrative graphics in Inkscape.

Take home message

0. Always aim for work quality suitable for collaboration!
1. Write tidy, well commented code. I.e.: "Will I understand my code completely by reading this 2 years from now?"
2. Write documentation! You'll be glad you did few years down the line.
3. Use version control, even if you are the only developer.
4. Try to cover with tests as much as possible.
5. Visualize whenever possible, develop fast pipelines for visualization.
6. Use multiple visualization, observe it from different angles before you decide how will you present it.
7. Every statement you write down requires evidence! I.e., data produced by you or references (scientific ones, not wikipedia).

Homework assignement

https://github.com/gzavo/CS_Assignment

1. Create a free github account if you don't already have one.
2. Fork this repository.
3. Create a markdown (.md) named "solution_.md" file that will contain the following:
4. The title of the following papers pivotal to our knowledge:

- MCC Van Dyke et al., 2019
- JT Harvey, Applied Ergonomics, 2002
- DW Ziegler et al., 2005

5. Create 1 plot from the dataset "istherecorrelation.csv", with DPI=300. The objective is to visualize the data as you see fit. Include the resulting image in the markdown file (and you can also write a few lines of interpretation if you like).

6. Commit and push these two files to your fork.
7. Create a pull request for me to this (original) repo. (Hint: use "compare across forks").

Thank you for your attention!

Questions?