Computing skills

Dr. Egidijus Kukstas

LIV.DAT advanced researcher skills school 2021/03/24

Overview

- Introduction: issues and goals
- Discussion of survey results
- Interacting with the Machine
- Data storage and management
- Software development and programming
- Break*
- Version control
- Multi-processing and GPU acceleration
- Presentation: making good plots
- Jobs and skills outside of academia

!!!
This is a group discussion and not 1-hour lecture. Feel free to contribute at any time!

*some work may be involved

The dangers of poor computing practices

- The Climate Research Unit (UEA) emails were hacked and released to the public in 2009
- “Climategate” ensued: scientists were manipulating the data – say conspiracy theorists
- No evidence of data manipulation was found
- In reality, poor data management and programming practices were highlighted
- Some emails contained discussion of bugs in the code
- Although the code was fine and the science checks out, the reputation was damaged
- First hand admissions from scientists: “Yup, my awful programming strikes again”
- With programming becoming an ever larger fraction of researchers’ activities, it is important to apply the same rigour as you do with publications
- Paper retractions due to mistakes happen all the time

https://www.nature.com/news/2010/101013/full/467775a.html

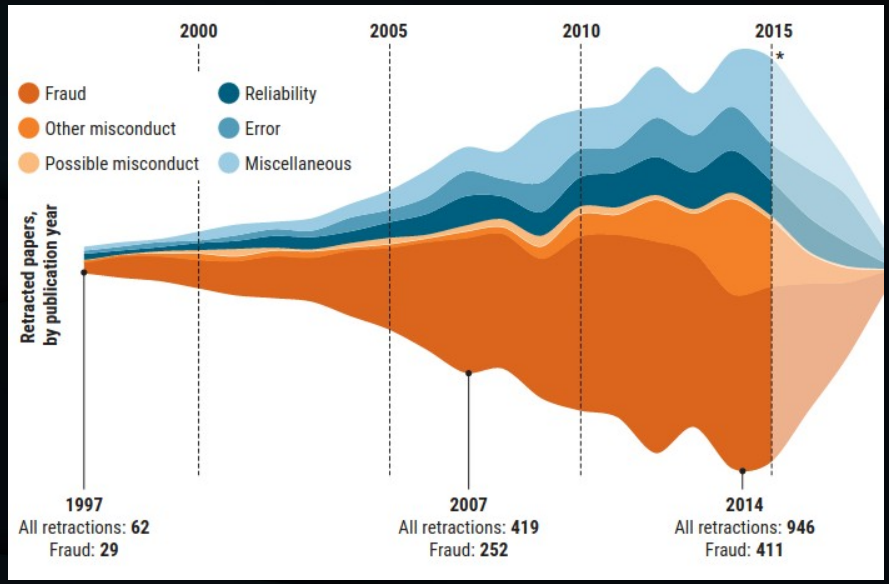


Image source: sciencemag.org
https://www.sciencemag.org/news/2018/10/what-massive-database-retracted-papers-reveals-about-science-publishing-s-death-penalty

Modern scientist vs. software developer

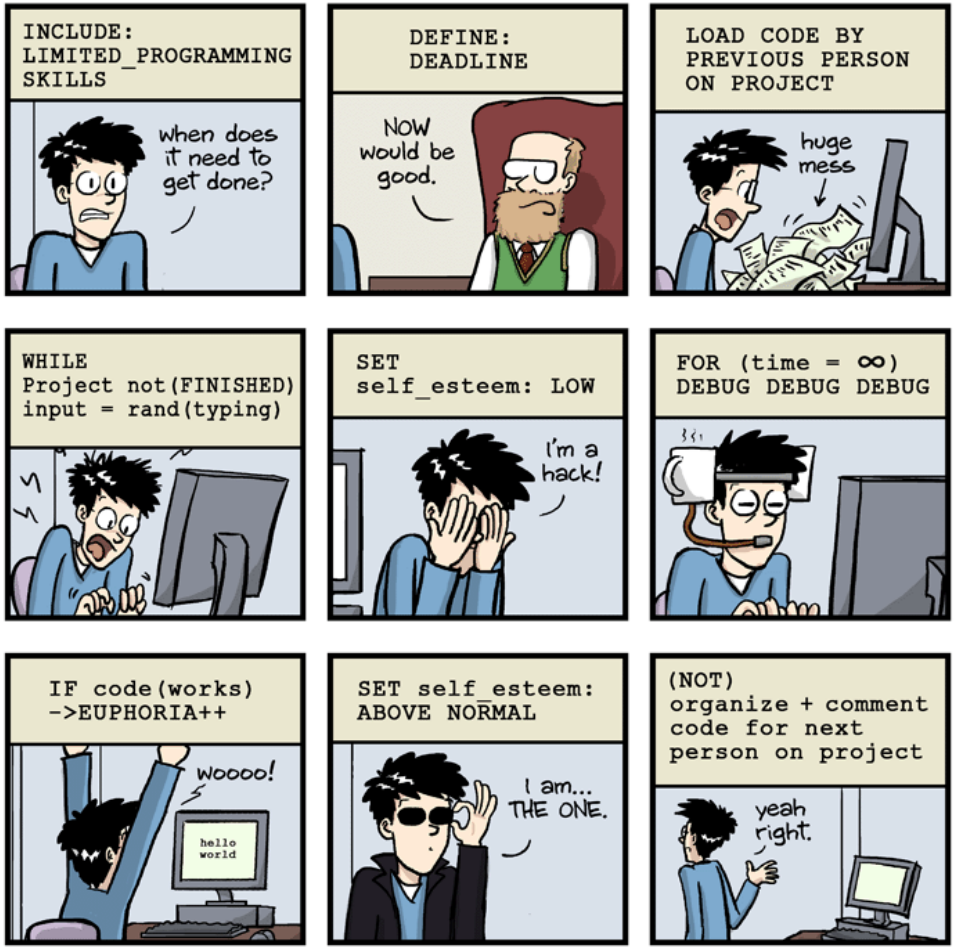
Issues

- Difference?
  - In science, software is viewed as a tool and not the product. Papers are.
- There is no code review. If the result matches our expectations then it's working correctly. Is it?
- Scientists are often self-taught (>90%)
- There is no reward for good code
- Projects grow organically with no clear requirements at the beginning
- Much of the code is never used again

Aims

- Software engineering/development has figured out many of the problems
- Practices exist that make code development faster, easier, better
- Can we learn something from them?
- Improve productivity, collaborate easier, have more confidence in results
- Mistakes in science can be extremely costly, how can we minimise them?
- Shake off the label of "bad programmer" if leaving academia

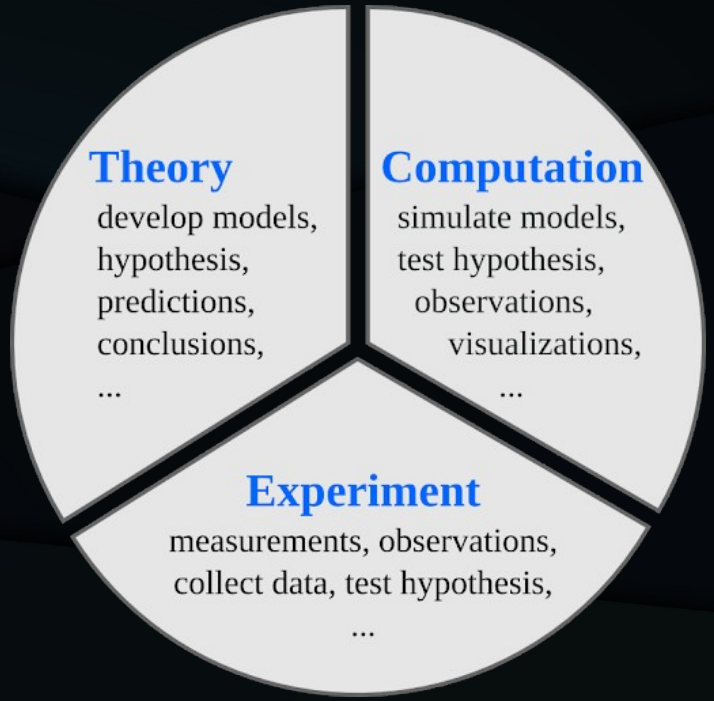
PROGRAMMING FOR NON-PROGRAMMERS



WWW.PHDCOMICS.COM

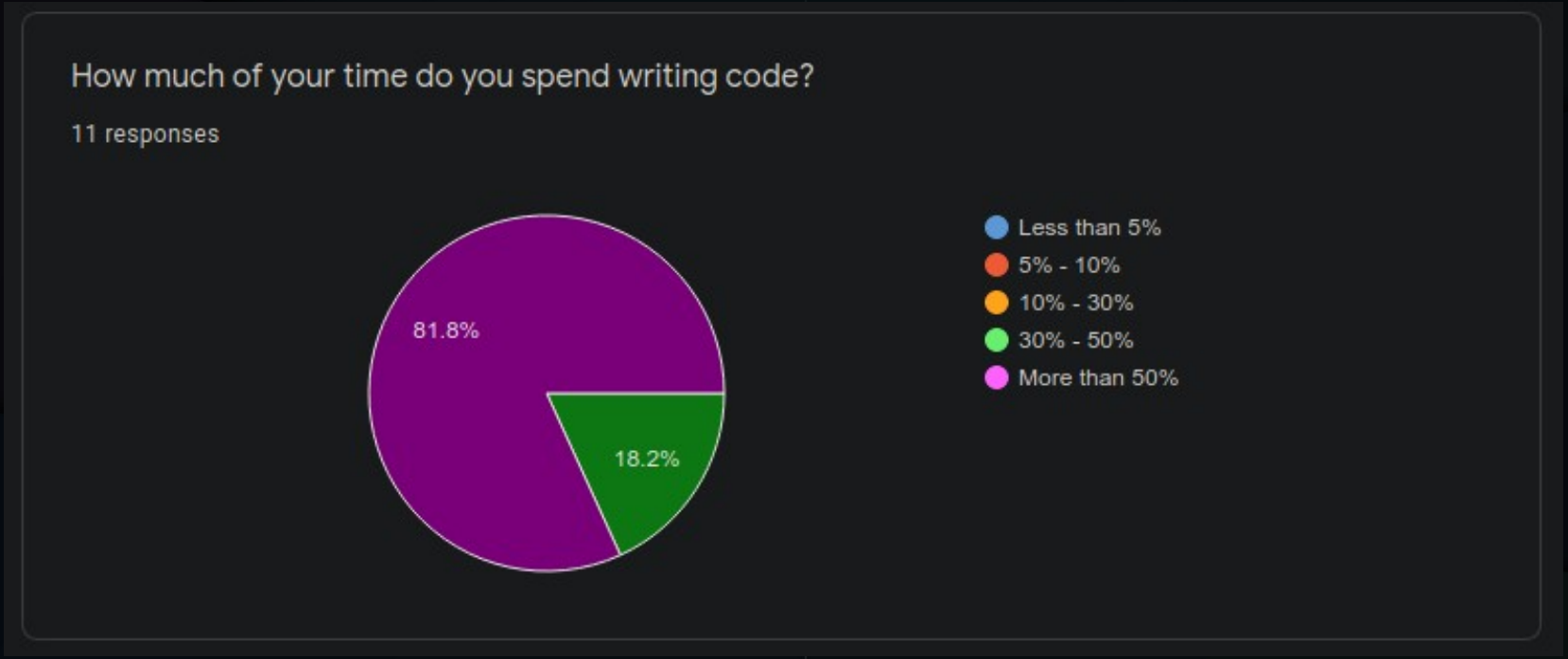
Goals of this workshop

- Identify the skills you already have but didn’t appreciate enough
- Identify the areas of computing which you are lacking in
- Gain awareness of best practice
  - The carpentry projects are good places to start*
- Have a collection of sources to refer to
- Find out what employers are looking for in industry



*https://datacarpentry.org/
https://software-carpentry.org/

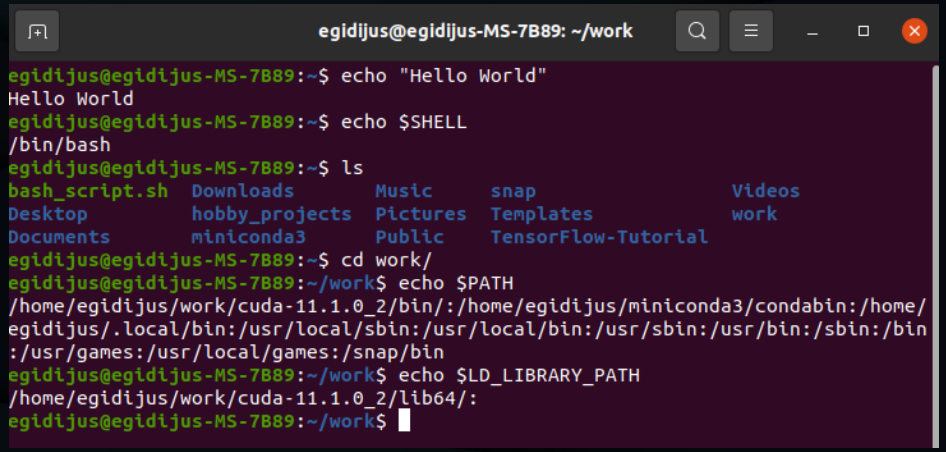
Survey results



https://forms.gle/vTBtwVF9Cigd4JuK9

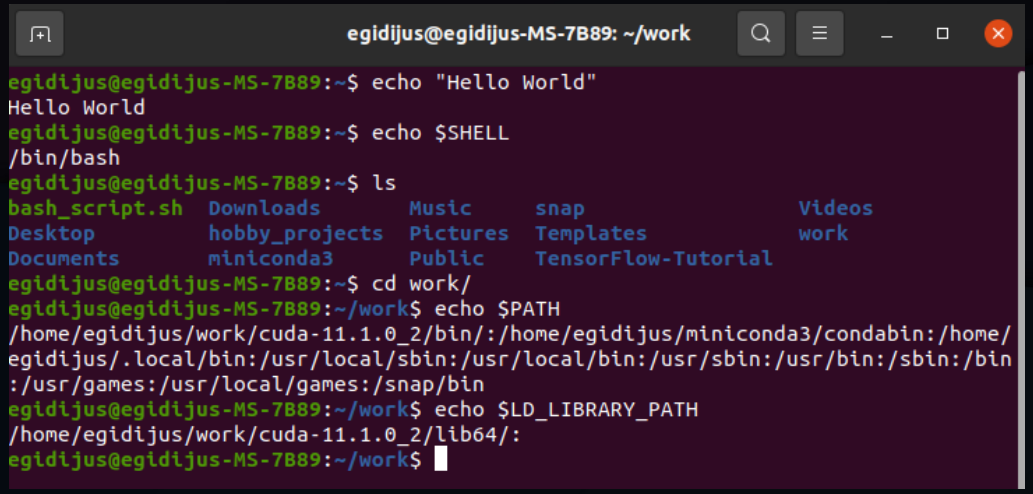
Interacting with the Machine

- Shell or “Terminal” is a program used to interact with the operating system
- Holds your commands in memory to be used again
- About 90% of the internet runs Linux servers
- AWS, Google cloud, and Azure all support Linux environments so the Shell is not going away anytime soon
- Various flavours: bash, csh, tcsh, ksh, etc.
- Simple use involves: ls, pwd, cd, cp, mv, etc.



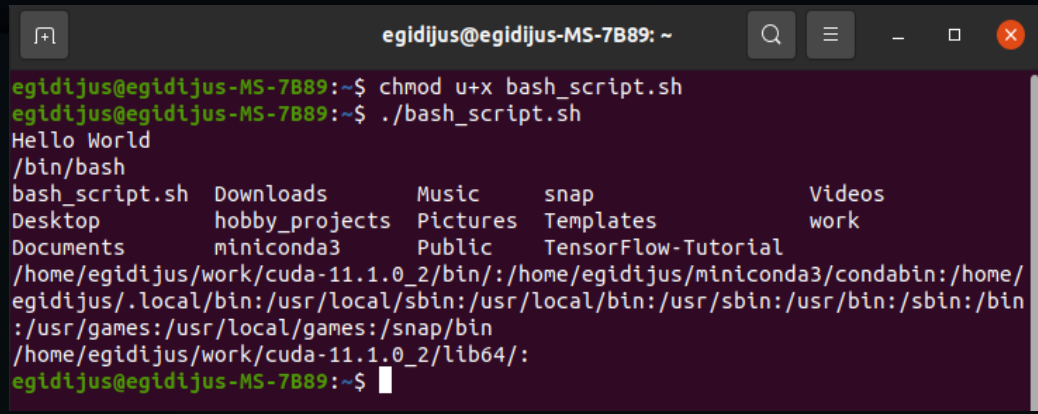
Shell scripts

You can type your commands one-by-one



Or you can save them in a file and execute them all

```
1 #!/bin/bash
2 echo "Hello World"
3 echo $SHELL
4 ls
5 cd work/
6
7 echo $PATH
8 echo $LD_LIBRARY_PATH
```



https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners

Environment variables

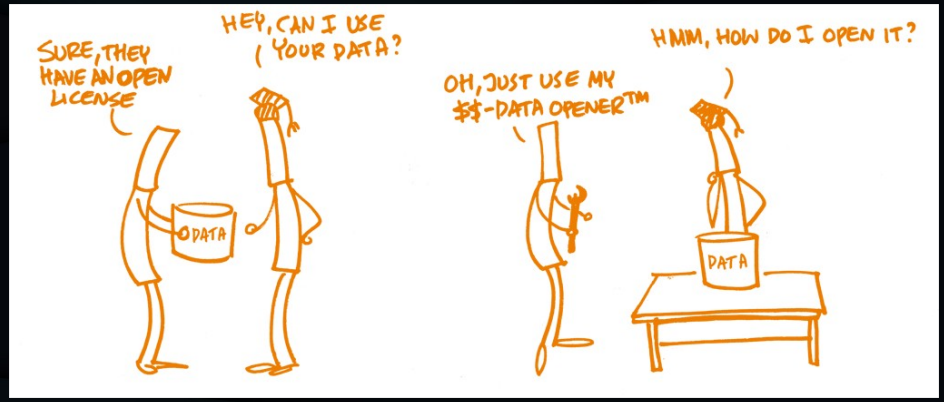
- Within Linux, global locations and values are stored in the form of environment variables to be looked up by shell applications
- Most commonly used are: PATH, LD_LIBRARY_PATH, PYTHONPATH, etc.
- How you control these depends on your shell
- They can be set once from the terminal or added to a file to be loaded at startup
- In my case, they should be stored in “~/.profile” or “~/.bashrc” if it doesn’t exist

```
egidijus@egidijus-MS-7B89:~$ export PATH="/home/egidijus/Music/:$PATH"
egidijus@egidijus-MS-7B89:~$ echo $PATH
/home/egidijus/Music/:/home/egidijus/work/cuda-11.1.0_2/bin/:/home/egidijus/mini
conda3/condabin:/home/egidijus/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sb
in:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
egidijus@egidijus-MS-7B89:~$ ls -a
.                   .dropbox        Public
..                  .dropbox-dist   .python_history
.apport-ignore.xml  .gnupg          snap
.bash_aliases       hobby_projects  .ssh
.bash_history       .ipython        .sudo_as_admin_successful
.bash_logout        .jupyter        Templates
.bashrc             .keras          TensorFlow-Tutorial
bash_script.sh      .kite           .thunderbird
.bogofilter         .local          Videos
.cache              miniconda3      .virtual_documents
.conda              .mozilla        .virtualenvs
.condarc            Music           .vscode
.config             .nv             .wget-hsts
Desktop             Pictures        work
.docker             .pki            .zoom
Documents           .profile
Downloads           .psensor
```

https://linuxize.com/post/how-to-set-and-list-environment-variables-in-linux/

Data storage and management: forms of good practice

- Save the raw data without any changes – preserve it for others to start where you did
- Save a version of the data that *you* want to use – change format, improve readability, save time
- Save data-point values for easy and fast plotting – you ***will*** have to remake that plot
- Favour open, non-proprietary format – CSV, JSON, HDF
- Record the steps taken to transform data and keep it with the data
- If data storage and manipulation is a significant part of your project, it may be worth reading up on data engineering



Wilson et al. 2016 “Good enough practices in scientific computing”

What format is best?

- There is no one “best” format for storing data
- This depends on your needs and resources
- Compressing the data can yield great size results at the cost of speed
- If you never plan to visually inspect the data, do you need it in human-readable format?
- Consider the overheads: CSV is quite efficient at storing small amounts of data*

storage	read_s	write_s	size_ratio_to_CSV
CSV	17.900	69.00	1.000
CSV.gzip	18.900	186.00	0.047
Pickle	0.173	1.77	0.374
HDF_fixed	0.196	2.03	0.435
HDF_tab	0.230	2.60	0.437
HDF_tab_zlib_c5	0.845	5.44	0.035
HDF_tab_zlib_c9	0.860	5.95	0.035
HDF_tab_bzip2_c5	2.500	36.50	0.011
HDF_tab_bzip2_c9	2.500	36.50	0.011

https://www.architecture-performance.fr/ap_blog/loading-data-into-a-pandas-dataframe-a-performance-study/

https://stackoverflow.com/a/37012035

Best practices in programming

Write programs for people, not computers

- Clear, short chunks of facts
  - A program should not require its readers to hold more than a handful of facts in memory at once
- Make names consistent, distinctive, and meaningful
- Make code style and formatting consistent
  - Will depend on your code language but most have a style guide

```
def create(path,archiveList,xFilesFactor=None,aggregationMethod=None,sparse=False,useFallocate=False):
  # Set default params
  if xFilesFactor is None:
    xFilesFactor = 0.5
  if aggregationMethod is None:
    aggregationMethod = 'average'

  #Validate archive configurations...
  validateArchiveList(archiveList)

  #Looks good, now we create the file and write the header
  if os.path.exists(path):
    raise InvalidConfiguration("File %s already exists!" % path)
  fh = None
  try:
    fh = open(path,'wb')
    if LOCK:
      fcntl.flock( fh.fileno(), fcntl.LOCK_EX )

    aggregationType = struct.pack( longFormat, aggregationMethodToType.get(aggregationMethod, 1) )
    oldest = max([secondsPerPoint * points for secondsPerPoint,points in archiveList])
    fh.write(packedMetadata)
    headerSize = metadataSize + (archiveInfoSize * len(archiveList))
    archiveOffsetPointer = headerSize

    for secondsPerPoint,points in archiveList:
      archiveInfo = struct.pack(archiveInfoFormat, archiveOffsetPointer, secondsPerPoint, points)
      fh.write(archiveInfo)
      archiveOffsetPointer += (points * pointSize)

    #If configured to use fallocate and capable of fallocate use that, else
    #attempt sparse if configure or zero pre-allocate if sparse isn't configured.
    if CAN_FALLOCATE and useFallocate:
      remaining = archiveOffsetPointer - headerSize
      fallocate(fh, headerSize, remaining)
    elif sparse:
      fh.seek(archiveOffsetPointer - 1)
      fh.write('\x00')
    else:
      remaining = archiveOffsetPointer - headerSize
      chunksize = 16384
      zeroes = '\x00' * chunksize
      while remaining > chunksize:
        fh.write(zeroes)
        remaining -= chunksize
      fh.write(zeroes[:remaining])

    if AUTOFLUSH:
      fh.flush()
      os.fsync(fh.fileno())
  finally:
    if fh:
      fh.close()
```

Wilson et al. 2014 “Best practices in scientific computing”

https://julien.danjou.info/python-bad-practice-concrete-case/

Let the computer do the work

- Make the computer repeat tasks: loops, functions
- Keep functions to doing one task at a time – easier to test
- Use existing modules/code rather than writing things from scratch
- Save recent commands in a file for reuse (write a script)
- If your work depends on repeated use of the same workflow: build it automatically
- Personal advice: DO NOT comment lines out as a way of controlling behaviour

This code would benefit so much from functions and loops

```
f = h5py.File('snapshot_011_z001p017_z1p0_paper_plot_quantities_10p0lMstar_30kpc_14p0lM200c_incBCG_field.hdf5', 'r')
SFR = np.array(f['SFR'])
fq_field = np.size(SFR[np.where(SFR < -12)])/np.size(SFR)
M_star_field = np.array(f['M_star'])

f = h5py.File('snapshot_011_z001p017_z1p0_paper_plot_quantities_10p0lMstar_30kpc_14p0lM200c_incBCG_cluster.hdf5', 'r')
SFR = np.array(f['SFR'])
fq_cluster = np.size(SFR[np.where(SFR < -12)])/np.size(SFR)
M_star_cluster = np.array(f['M_star'])
```

Forgetting to rename on of these instances is only a matter of time

Make incremental changes

- Work in small steps with frequent feedback and course correction
  - Really helps if you structure your code well (small functions)
- Use a version control system
  - You rarely know if the changes you make will be positive
  - More on version control later

- One example of incremental development may be:
  - Pseudocode draft
  - Define variables you know you are going to need, in the format you think you need them
  - Empty functions for functionality
  - Finalise function logic, returning intermediate values to test functionality
  - Test final function with various inputs
  - Add comments and optimise if possible

https://dfrieds.com/python/incremental-development.html

Don’t repeat yourself or others

- Data input, constants, variable values must have a single, authoritative source
- Store parameter values in a separate file if you have many
- Modularise, rather than copy/paste
  - OOP may come into play here
- Re-use and don’t re-write
  - On a wider scale: modules
  - On a small scale: worth the time searching for that bit of code you wrote in the past
- Plan for mistakes
  - functions/classes are much easier to unit-test
  - Implement assertions (check on inputs, format, etc.)

```
11  def compute_fq_from_snapshot(filename):
12      '''
13      Inputs: (string) file name
14      Computes quenched fraction given a snapshot file name
15      Outputs: (tuple) quenched fraction, stellar mass
16      '''
17      f = h5py.File(filename, 'r')
18      SFR = np.array(f['SFR'])
19      M_star = np.array(f['M_star'])
20      fq = np.size(SFR[np.where(SFR < -12)])/np.size(SFR)
21      return fq, M_star
```

Optimise software only after it works correctly

- Clean up comments, general tidy up
- Use a profiler to find bottlenecks
- Consider parallelising if it takes a while to run
- Write in the highest level language possible, then translate to something more efficient if needed
  - e.g. start in python and translate to C/FORTRAN
  - The overall structure is unlikely to change
  - Evidence shows that you work faster in high level language or one you are familiar with

Oram, A. 2010 “Two comparisons of programming languages”

```python
import time
import concurrent.futures
from PIL import Image, ImageFilter

img_names = [
    'photo-1516117172878-fd2c41f4a759.jpg',
    'photo-1532009324734-20a7a5813719.jpg',
    'photo-1524429656589-6633a470097c.jpg',
    'photo-1530224264768-7ff8c1789d79.jpg',
    'photo-1564135624576-c5c88640f235.jpg',
    'photo-1541698444083-023c97d3f4b6.jpg',
    'photo-1522364723953-452d3431c267.jpg',
    'photo-1513938709626-033611b8cc03.jpg',
    'photo-1507143550189-fed454f93097.jpg',
    'photo-1493976040374-85c8e12f0c0e.jpg',
    'photo-1504198453319-5ce911bafcde.jpg',
    'photo-1530122037265-a5f1f91d3b99.jpg',
    'photo-1516972810927-80185027ca84.jpg',
    'photo-1550439062-609e1531270e.jpg',
    'photo-1549692520-acc6669e2f0c.jpg'
]

t1 = time.perf_counter()

size = (1200, 1200)


def process_image(img_name):
    img = Image.open(img_name)

    img = img.filter(ImageFilter.GaussianBlur(15))

    img.thumbnail(size)
    img.save(f'processed/{img_name}')
    print(f'{img_name} was processed...')


with concurrent.futures.ProcessPoolExecutor() as executor:
    executor.map(process_image, img_names)


t2 = time.perf_counter()

print(f'Finished in {t2-t1} seconds')
```

Document design and purpose, not mechanics

- Some comments is usually better than no comments at all
- Proper, meaningful comments is what really makes a difference
- Document the purpose of the code rather than what it does – this should be conveyed by the code itself
- Refactor code if it helps explain how it works
- Embed documentation within the code itself
- If excessive documentation is required, consider a format specifically designed for it, e.g. Jupyter notebooks

No comments at all

```
r = n / 2;
while ( abs( r - (n/r) ) > t ) {
    r = 0.5 * ( r + (n/r) );
}
System.out.println( "r = " + r );
```

Good and unnecessary comments

```
// square root of n with Newton-Raphson approximation
r = n / 2;
while ( abs( r - (n/r) ) > t ) { // while result is greater than tolerance
    r = 0.5 * ( r + (n/r) );
}
System.out.println( "r = " + r );
```

Descriptive names instead of comments

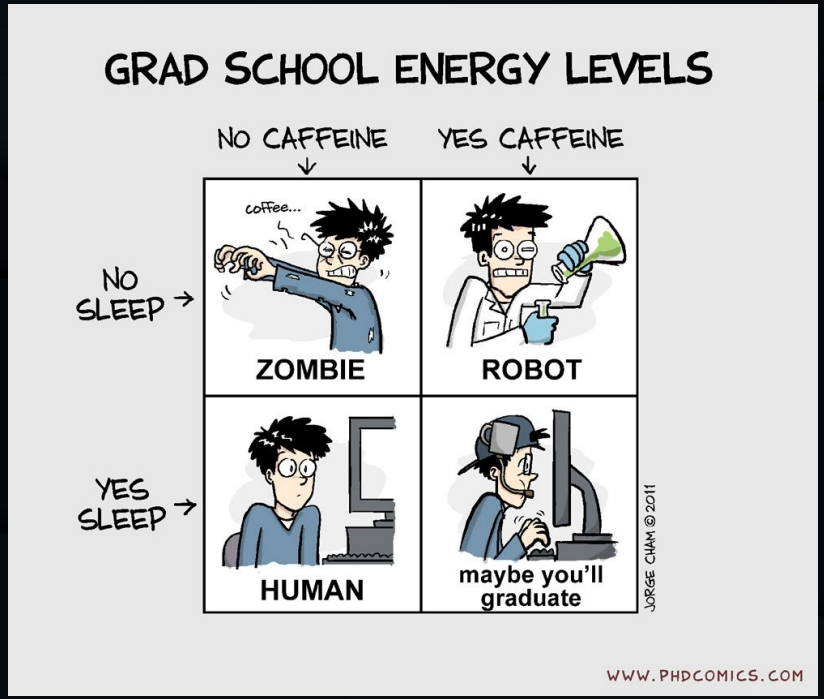
```
private double SquareRootApproximation(n) {
    r = n / 2;
    while ( abs( r - (n/r) ) > t ) {
        r = 0.5 * ( r + (n/r) );
    }
  return r;
}
System.out.println( "r = " + SquareRootApproximation(r) );
```

https://blog.codinghorror.com/coding-without-comments/

A working break

Task: find a job you would consider outside of academia, list:

- job title,
- website you found it on,
- one key technical skill it requires
- https://docs.google.com/document/d/1mRyHjTynmQFqVC2W-IxIUXYZVKOI8-FgRIiIXGyhqbg/edit?usp=sharing



Version control: Git

- Git stores metadata on your code and tracks changes
- Dated “images” of your code are available to access should something go wrong
- Github, Gitlab, Gitbucket, etc. are online repositories where your code can be stored and shared with others
  - This allows for collaborative development where other people can propose changes/additions to your code
  - You then get to decide whether to “merge” the changes or not
- Git is very powerful and not always intuitive
- Practical advice: at the very least, commit your code regularly as a form of backup for yourself, whatever the state

```
egidijus@egidijus-MS-7B89:~/work/projects/ctr_ml/analysis$ git init
Reinitialised existing Git repository in /home/egidijus/work/projects/ctr_ml/analysis/.git/
egidijus@egidijus-MS-7B89:~/work/projects/ctr_ml/analysis$ git add *
egidijus@egidijus-MS-7B89:~/work/projects/ctr_ml/analysis$ git commit -m "Initial commit"
[master 8958c7c] Initial commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test_file
egidijus@egidijus-MS-7B89:~/work/projects/ctr_ml/analysis$ git remote add origin https://github.com/EKukstas
/work_uol.git
fatal: remote origin already exists.
egidijus@egidijus-MS-7B89:~/work/projects/ctr_ml/analysis$ git push -u origin master
Username for 'https://github.com': ^C
egidijus@egidijus-MS-7B89:~/work/projects/ctr_ml/analysis$
```

https://rogerdudler.github.io/git-guide/

Parallel computing

- Most CPUs these days have multiple cores
- Scientific computing clusters have 100s or even 1000s of cores
- Most processes are designed to only use a single core
  - Explicit instructions have to be passed in or with your code to use multiple cores
- It is quite easy to implement and can speed up your code multiple times over
- Specific tasks, such as machine learning can take advantage of the GPU, as well
  - You will need to install specific drivers and set your code up to use it, which can be more involved
  - The rewards can be big if your problem is well suited (matrix operations, etc.)

```python
img_names = [
    'photo-1516117172878-fd2c41f4a759.jpg',
    'photo-1532009324734-20a7a5813719.jpg',
    'photo-1524429656589-6633a470097c.jpg',
    'photo-1530224264768-7ff8c1789d79.jpg',
    'photo-1564135624576-c5c88640f235.jpg',
    'photo-1541698444083-023c97d3f4b6.jpg',
    'photo-1522364723953-452d3431c267.jpg',
    'photo-1513938709626-033611b8cc03.jpg',
    'photo-1507143550189-fed454f93097.jpg',
    'photo-1493976040374-85c8e12f0c0e.jpg',
    'photo-1504198453319-5ce911bafcde.jpg',
    'photo-1530122037265-a5f1f91d3b99.jpg',
    'photo-1516972810927-80185027ca84.jpg',
    'photo-1550439062-609e1531270e.jpg',
    'photo-1549692520-acc6669e2f0c.jpg'
]

t1 = time.perf_counter()

size = (1200, 1200)


def process_image(img_name):
    img = Image.open(img_name)

    img = img.filter(ImageFilter.GaussianBlur(15))

    img.thumbnail(size)
    img.save(f'processed/{img_name}')
    print(f'{img_name} was processed...')

for img in img_names:
    process_image(img)

t2 = time.perf_counter()

print(f'Finished in {t2-t1} seconds')
```

```
Finished in 12.15791793000244 seconds
```

```python
img_names = [
    'photo-1516117172878-fd2c41f4a759.jpg',
    'photo-1532009324734-20a7a5813719.jpg',
    'photo-1524429656589-6633a470097c.jpg',
    'photo-1530224264768-7ff8c1789d79.jpg',
    'photo-1564135624576-c5c88640f235.jpg',
    'photo-1541698444083-023c97d3f4b6.jpg',
    'photo-1522364723953-452d3431c267.jpg',
    'photo-1513938709626-033611b8cc03.jpg',
    'photo-1507143550189-fed454f93097.jpg',
    'photo-1493976040374-85c8e12f0c0e.jpg',
    'photo-1504198453319-5ce911bafcde.jpg',
    'photo-1530122037265-a5f1f91d3b99.jpg',
    'photo-1516972810927-80185027ca84.jpg',
    'photo-1550439062-609e1531270e.jpg',
    'photo-1549692520-acc6669e2f0c.jpg'
]

t1 = time.perf_counter()

size = (1200, 1200)
    Find related code in corey_schafer_code_snippets

def process_image(img_name):
    img = Image.open(img_name)

    img = img.filter(ImageFilter.GaussianBlur(15))

    img.thumbnail(size)
    img.save(f'processed/{img_name}')
    print(f'{img_name} was processed...')


with concurrent.futures.ProcessPoolExecutor() as executor:
    executor.map(process_image, img_names)


t2 = time.perf_counter()

print(f'Finished in {t2-t1} seconds')
```

```
Finished in 2.2747356240033696 seconds
```

https://youtu.be/fKl2JW_qrso
http://bit.ly/multiprocess-code

Presentations

- Use whatever tool you like, but make sure it works the same way on any system
  - re. PowerPoint and LibreOffice Impress
  - PDF is a good format
- Plots are the ultimate product – make sure they are of highest quality
  - Frequently presented at talks / on posters by other people if they're good
- Make sure that:
  - The message is as clear as possible without a lengthy caption (out of context colleague test)
  - Labels are clear and large enough
  - Consider colour blindness: choose your colours carefully and favour line-types over different colours
  - Favour vector images over raster
  - Plot contours instead of many scatter points, rasterize if absolutely necessary
- Avoid plots from simulation packages – you have little control and they're usually poor. Save the data instead and plot it yourself
- Python, R, Matlab all have good plotting modules where you have full control

Example parameter changes for Python's matplotlib

```
fsize = 15
tsize = 18

tdir = 'in'

major = 5.0
minor = 3.0

lwidth = 0.8
lhandle = 2.0

plt.style.use('default')
plt.rcParams['text.usetex'] = True
plt.rcParams['font.size'] = fsize
plt.rcParams['legend.fontsize'] = tsize
plt.rcParams['xtick.direction'] = tdir
plt.rcParams['ytick.direction'] = tdir
plt.rcParams['xtick.major.size'] = major
plt.rcParams['xtick.minor.size'] = minor
plt.rcParams['ytick.major.size'] = 5.0
plt.rcParams['ytick.minor.size'] = 3.0
plt.rcParams['axes.linewidth'] = lwidth
plt.rcParams['legend.handlelength'] = lhandle
```

https://towardsdatascience.com/matplotlib-styles-for-scientific-plotting-d023f74515b4

Concluding remarks

- Appreciate the importance of programming in your professional lives
  - Reproducibility, accountability depend on it
- Continue to improve: look at your code from start of the PhD
- Don’t expect to implement all of the advice now
- Be aware of what good practices are
- It will help you in the long run