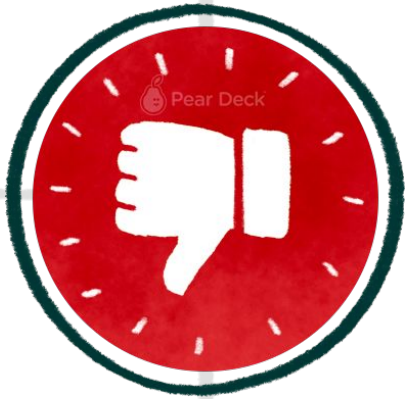
HARLEY WOOD SCHOOL 2019

Good Code in practice

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

Drag your dot to show if you're ready to move on:



What version of Python do you use (the most)?

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

Python 2.7

print “End of Life”

5/2 = 2

Security and bug fixes only

No new features

Many modules will drop python2 support in current/future versions

from future import print_function, division

Python 3

print(“Current and future”)

5/2 = 2.5

New and current modules will only support python3

Convert **your** code with helper tools and a basic tutorial.

How would you feel showing your code to a peer?

Students, drag the icon!

How would you feel showing your code to a **supervisor**?

How would you feel showing your code to the world?

P.S.. there is a **LOT** of shitty code on the internet.
Your +1 isn't going to end the world, and it'll get better over time.

The Zen of Python

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

>> import this

Create readable code

ASTRONOMY DATA AND COMPUTING SERVICES

Python was designed to be readable

Code-blocks are defined by indentation

Line continuations are not required

Syntax is human readable

```
a="""Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
"""

lines = a.split('\n') # \n is the newline character
num_lines = len(lines)

nwords = 0
for line in lines:
    words = line.split()
    nwords += len(words)
```

Readability

ASTRONOMY DATA AND COMPUTING SERVICES

For dictionaries, lists, sets and tuples use multiple lines to be more clear

Inline comments are allowed

```
fmts = {
        "ann": "Kvis annotation",
        "reg": "DS9 regions file",
        "fits": "FITS Binary Table",
        "csv": "Comma separated values",
        "tab": "tabe separated values",
        "tex": "LaTeX table format",
        "html": "HTML table",
        "vot": "VO-Table",
        "xml": "VO-Table",
        "db": "Sqlite3 database",
        "sqlite": "Sqlite3 database"}
```

```
my_matrix = [[1, 0, 1],
             [0, 1, 0],
             [1, 0, 0]]
```

```
books = ['Advanced Engineering Maths, Kreyszig', # Standard textbook
         # 'Flowers for Algernon, Keyes', # Great read
         'Rendezvous with Rama, Clarke', # Classic Sci-fi
        ]
```

Multi lines for function def/call

ASTRONOMY DATA AND COMPUTING SERVICES

```
def do_lots_of_things(option1, # required
                      option2=0, # not required, has default value
                      option3=1,
                      labels=(),
                      sqlconnecion=None,
                      retries=3,
                      verbose=False,
                      do_print=True):
    pass
```

```
do_lots_of_things('all the things',
                  labels=('one','two','three'),
                  verbose=True, do_print=False)
```

Indenting

ASTRONOMY DATA AND COMPUTING SERVICES

Python uses indenting to identify code blocks.

Advantage: No need for bracers, or semi-colons, easily readable

Disadvantage: Python allows **tabs** or **spaces** or some **mix** of the two
[worst design choice ever made IMO]

Solution: Be **consistent** with indenting, ideally use 4 spaces (but 2 is ok)

Indenting issues

```
def function(firstArgument,
             secondOne,
             modified);
```

What you **see**
(In this editor anyway)

```
def function(firstArgument,
→ → → → ·····secondOne,
→ → → ·······modified);
```

What you **have**

Many editors will bind the “tab” key to the “tab-ify” function.
Eg your Jupyter notebook

Interactive Development Environment

ASTRONOMY DATA AND COMPUTING SERVICES

And IDE is like a text editor but with lots of extra fancy-ness added on.

In fact you can take your favorite text editor (emacs or vim) and give it an upgrade with plugins that will turn it into more of an IDE.

Syntax Highlighting and Checking

Auto Indentation

Spell Checking (language aware)

Get a ‘real’ IDE

ASTRONOMY DATA AND COMPUTING SERVICES

Includes: debugging tools, integration with version control, refactoring tools, templates for new modules/files and docstrings.

PyCharm (not just for python)

Spyder (part of anaconda install)

Eclipse

Sublime Text

What kind of IDE do you usually use?

Full featured.
With Git, Templates, etc.

Some features.
Colours and highlighting.

Basic text editor.
No added value.

Students, drag the icon!

Did you know that many IDEs are free for academic use?

‘One-liners’ are write-only

ASTRONOMY DATA AND COMPUTING SERVICES

What does this code do?

How does it do it?

```
print(reduce( (lambda r,x: (r.difference_update(range(x*x,N,2*x)) or r)
                     if (x in r) else r),
        range(3, int((N+1)**0.5+1), 2),
        set([1,2] + range(3,N,2))))
```

‘One-liners’ are write-only

ASTRONOMY DATA AND COMPUTING SERVICES

What does this code do?

How does it do it?

```
print(reduce( (lambda r,x: (r.difference_update(range(x*x,N,2*x)) or r)
                     if (x in r) else r),
        range(3, int((N+1)**0.5+1), 2),
        set([1,2] + range(3,N,2))))
```

```
set([1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97])
```

Readability counts

ASTRONOMY DATA AND COMPUTING SERVICES

What does this code do, and how?

```
from math import sqrt
N = 100
sieve = list(range(N))
for number in range(2, int(sqrt(N))+1):
    if sieve[number]:
        for multiple in range(number**2,N,number):
            sieve[multiple] = False

for number, prime in enumerate(sieve):
    if prime:
        print(number)
```

Two unsolved problems in Computing

ASTRONOMY DATA AND COMPUTING SERVICES

1. Cache invalidation
2. Naming things
3. Off-by-one errors

Two unsolved problems in Computing

ASTRONOMY DATA AND COMPUTING SERVICES

1. We have only one joke
2. It’s not funny

Naming conventions

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

Use words!

Preferably: nouns for classes and variables, verbs for functions, (adjectives for decorators?)

Be verbose but not needlessly so.

underscores_for_functions

CamelCaseForClasses

ALL_CAPS_FOR_STATIC_VARIABLES

Naming conventions

ASTRONOMY DATA AND COMPUTING SERVICES

Python doesn’t enforce static/private variables or functions.

However, in practice:

STATIC_VARIABLE

_private_function_or_variable (not part of a public API)

Naming conventions

ASTRONOMY DATA AND COMPUTING SERVICES

Loop iterators are commonly just **i, j, k** which is not very descriptive.

Use names.

```
from math import sqrt
N = 100
sieve = list(range(N))
for number in range(2, int(sqrt(N))+1):
    if sieve[number]:
        for multiple in range(number**2,N,number):
            sieve[multiple] = False

for number, prime in enumerate(sieve):
    if prime:
        print(number)
```

Python is (too) flexible

ASTRONOMY DATA AND COMPUTING SERVICES

Python lets you redefine just about everything, even though you shouldn't:

dir(__builtins__) are all functions or variables that you can, **but shouldn't**, redefine.

Annoying ones are:

buffer, dir, eval, exit, **file**, format, help, **id**, **input**, len, **map**, max, min, next, object, open, range, **type**, zip

If you assign a value to one of these then you can get some very hard to debug side-effects.

The “I don’t care about this” name

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

_ always contains the value of the last evaluated statement

It can be used to absorb junk you don’t care about:

```
mean, _, number = function_returns_three_things()
```

People know to ignore anything that is chucked into _

Think of it kind of like /dev/null

Match the variable name with its intended use:

i

cost

len

begin

j

cmap

The total cost of items in a list

The index of the array where we begin a search

The length of an observation

Row index

Column index

Color map (object)

Students, draw anywhere on this slide!

Who likes making cakes?

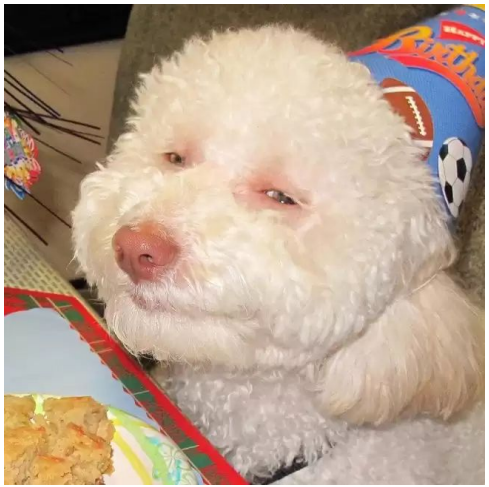
Code Layout

ASTRONOMY DATA AND COMPUTING SERVICES

Writing code is not a story that unfolds and entertains people with twists and character developments.

It's a **recipe**. Like for yummy cakes.

1. Ingredients for the shopping list ⇒ modules to import
2. Description of techniques ⇒ functions
3. Directions ⇒ code in main scope



Template for code layout

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

- Preamble
- Imports
- Static and global variables (be judicious)
- Classes
- Functions
- If __main__

Structure.ipynb

ASTRONOMY DATA AND COMPUTING SERVICES

Do the thing.

Modules

ASTRONOMY DATA AND COMPUTING SERVICES

Use modules if you have code common to multiple programs.

Use modules if your code has become very long and hard to navigate within a file.

Use and if __main__ clause to stop code from executing when imported as a module.

Simple module:

- mkdir mymodule
- touch mymodule/__init__.py
- cp code.py mymodule/.
- >>> from mymodule import code

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

Give me an example of an idiom:

Students, write your response!

Idioms

ASTRONOMY DATA AND COMPUTING SERVICES

Beating around the bush.

Cutting corners.

Get all your ducks in a row.

Under the weather.

Once you know them they are good shorthands.

Idiomatic Python

ASTRONOMY DATA AND COMPUTING SERVICES

There are common ways of doing common tasks, we call these idioms.

By following idioms you reduce your risk of bugs, and make it easier for others to follow your work.

https://gist.github.com/dpallot/1aadff223f3b3efbec8e is a good place to start

Leap before you look

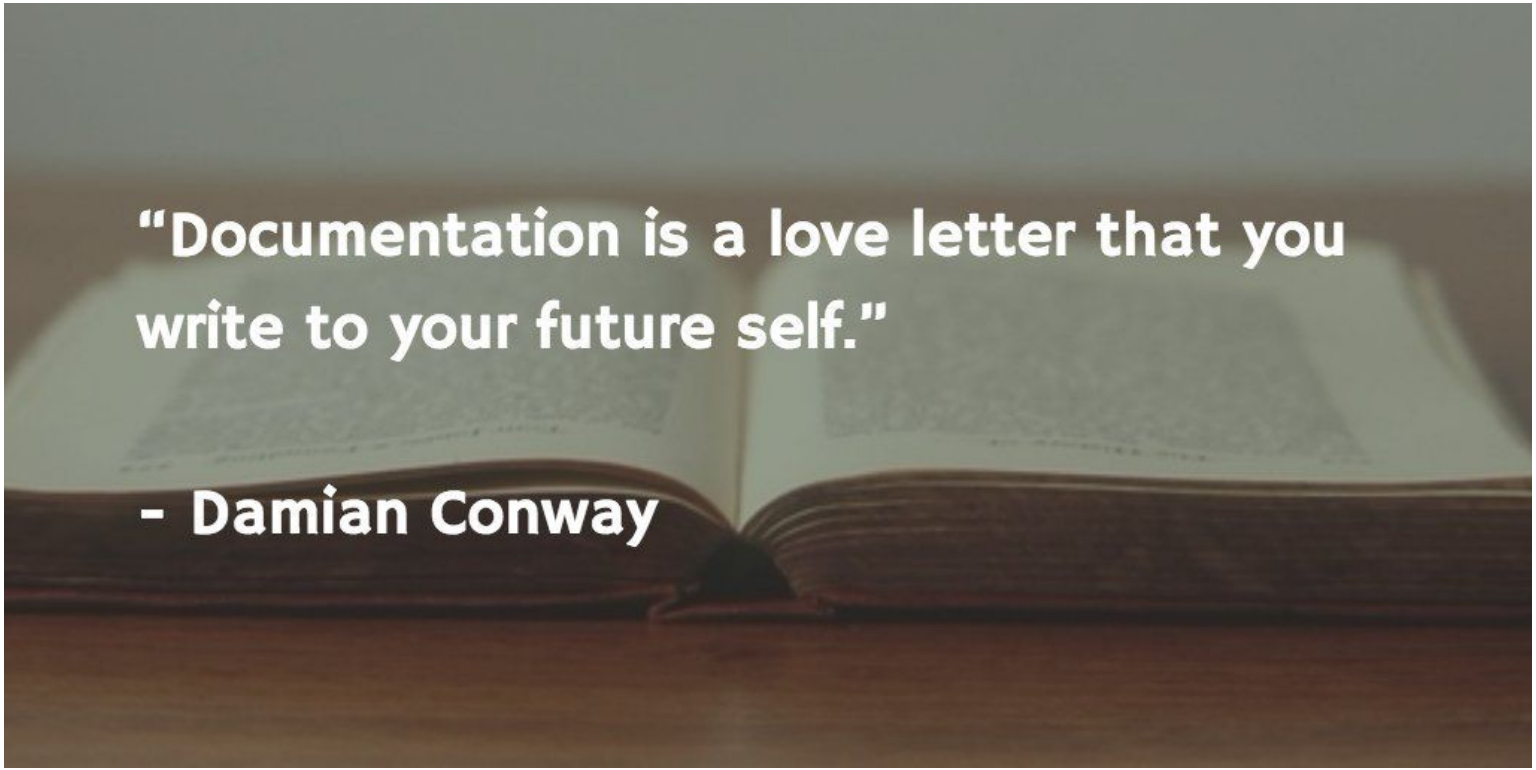
Throw caution to the wind, clean up and apologize when it doesn't work.

Exceptions.ipynb

```
try:
    #do something you hope will work
    undefined_function()
except NameError as e:
    # Apologize and clean up
    print("Function doesn't exist?")
    # raise e
except ValueError as e:
    print("Function defined but it's borked somehow.")
else:
    # Run if there were no errors at all
    print("That worked as planned!")
finally:
    # Run reguardless of errors or not
    print("We are finally done, pack it all away.")
print("Why do we need finally?")
```

Document your work

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES



Comments are not documentation

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

Documentation is for people **using** the code (regular folks)

Documentation describes the ingredients and what kind of sausages are made.

Comments are for people **reading** the code (ie developers and future you)

Comments are about the sausage making process (ew!)

```
def galactic2fk5(l, b):
    """
    Convert galactic l/b to fk5 ra/dec

    Parameters
    ----------
    l, b : float
        Galactic coordinates in radians.

    Returns
    -------
    ra, dec : float
        FK5 ecliptic coordinates in radians.
    """
    a = SkyCoord(l, b, unit=(u.radian, u.radian), frame='galactic')
    return a.fk5.ra.radian, a.fk5.dec.radian
```

Docstrings

ASTRONOMY DATA AND COMPUTING SERVICES

Triple quoted text immediately after a class or function definition is a docstring.

Any format is fine, though some conventions make it easier to work with.

Docstring: epytext (or javadoc)

ASTRONOMY DATA AND COMPUTING SERVICES

```
def epytext_func(param1, param2):
    """
    This is a epytext style.

    @param param1: this is a first param
    @param param2: this is a second param
    @return: this is a description of what is returned
    @raise keyError: raises an exception
    """
    return
```

Docstring: reStructuredText (reST)

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

```
def rest_func(param1, param2):
    """
    This is a reST style.

    :param param1: this is a first param
    :param param2: this is a second param
    :returns: this is a description of what is returned
    :raises keyError: raises an exception
    """
    return
```

PEP 287 says this is the prefered style for python documentation

Docstring: numpydoc

Much more flexible and has many more features than other docstring formats.

Used by numpy, scipy, astropy, etc.

```
def numpydoc_func(first, second):
    """
    My numpydoc description of a kind
    of very exhautive numpydoc format docstring.

    Parameters
    ----------
    first : array_like
        the 1st param name `first`
    second :
        the 2nd param
    third : {'value', 'other'}, optional
        the 3rd param, by default 'value'

    Returns
    -------
    string
        a value in a string

    Raises
    ------
    KeyError
        when a key error
    OtherError
        when an other error
    """
    return
```

IDE and Docstrings

ASTRONOMY DATA AND COMPUTING SERVICES

Docstrings can help you while **you** develop code because many IDE's recognize docstring formats.

They can give tooltips, autocompletion, and note errors, as you code, before you even run the program.

This is a nice way to avoid bugs!

DRY or DIE!

ASTRONOMY DATA AND COMPUTING SERVICES

Don’t Repeat Yourself (Duplication Is Evil)

Duplicated code means duplicated errors and bugs

Write a function, call it many times

Better still, write a module

DRY examples.ipynb

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

Do the thing!

The DRY principle - II (or DRO maybe?)

ASTRONOMY DATA AND COMPUTING SERVICES

Don’t Repeat Others

- (re-) implementing code often means going through the same growth/development curve of bugs and corner cases
- Common problems have common solutions, use them!
- ‘import’ your way to success

DRO examples

ASTRONOMY DATA AND COMPUTING SERVICES

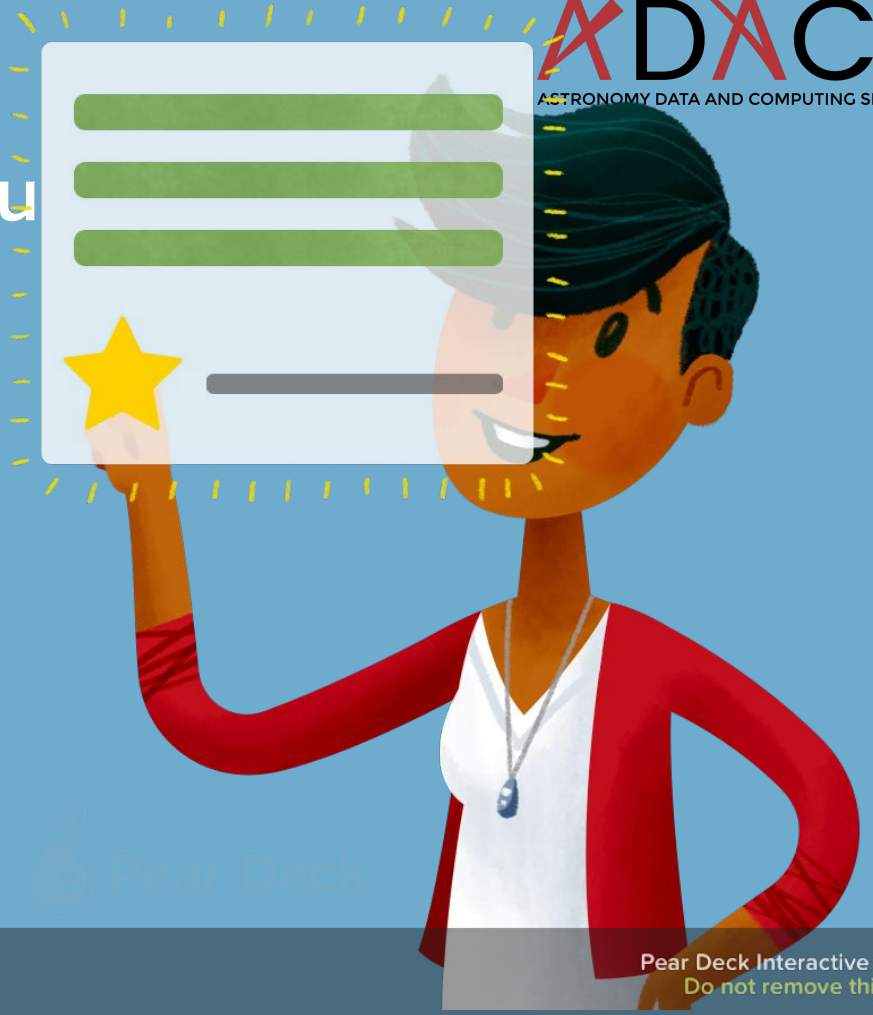
The following are my ‘go-to’ list of modules that i’ll usually load for a new project before I write any code at all, because I’m 99% likely to use them.

astropy, numpy, scipy, matplotlib, pandas, [ephem, skyfield]

Name a module that you haven't used but would like to know more about.

- or -

Describe a module you wish existed.



Never do ‘from xxx import *’

ASTRONOMY DATA AND COMPUTING SERVICES

Both math and numpy provide trig functions, but somehow this breaks… Why?

```
from math import *
from numpy import *

print(cos([0,0.5,1])) # prints [ 1.          0.87758256  0.54030231]
print(acos(cos([0,0.5,1]))) # Raises TypeError !?
```

Namespaces

ASTRONOMY DATA AND COMPUTING SERVICES

Namespaces help avoid name collisions, keeps track of where functions/variables come from.

```
import math
import numpy as np # common shorthand

print(np.cos([0, 0.5, 1])) #prints [ 1.          0.87758256  0.54030231]

try:
    # same error as before but now we have some hint why it occurs!
    print(math.acos(np.cos([0, 0.5, 1])))
except TypeError:
    print("It broke, lets try using map()")
    print(map(math.acos, np.cos([0, 0.5, 1])))
```

Separate code and data

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

Having a script that needs to be edited every time it runs is just asking for trouble.

Let’s look at KeepThemSeparated.ipynb

Test code

ASTRONOMY DATA AND COMPUTING SERVICES

The only thing that people write less than documentation is test code.

Pro-tip: Both documentation and test code is easier to write if you do it as part of the development process.

1. Write function definition and basic docstring
2. Write function contents
3. Write test to ensure that function does what the docstring claims.
4. Update code and/or docstring until (3) is true.

What to test?

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

Whatever you currently do to convince yourself that your code works is a test!

Everytime you find a but or some corner case, write a test that will check it.

Making mistakes doesn't make you a bad person,
making the **same mistake** over and over does.

Where to test?

ASTRONOMY DATA AND COMPUTING SERVICES

Your if __main__ clause is a great place to kick off tests.

For script.py consider making test_script.py which imports script.py and tests all the functions therein.

Useful bit of magic (that doesn't abide our rules of goodness):

```
if __name__ == "__main__":
    # introspect and run all the functions starting with 'test'
    for f in dir():
        if f.startswith('test'):
            print(f)
            globals()[f]()
```

Testing.ipynb

ASTRONOMY DATA AND COMPUTING SERVICES

Do the thing.

Vectorizing code

ASTRONOMY DATA AND COMPUTING SERVICES

Functions that work on single values can be made to work on lists, sets, arrays and just about any iterable object. This is called vectorizing code.

Vectorizing is super useful when paired with numpy arrays and related functions as they have been optimised for this purpose.

Vectorize.ipynb

ASTRONOMY DATA AND COMPUTING SERVICES

Do the thing.

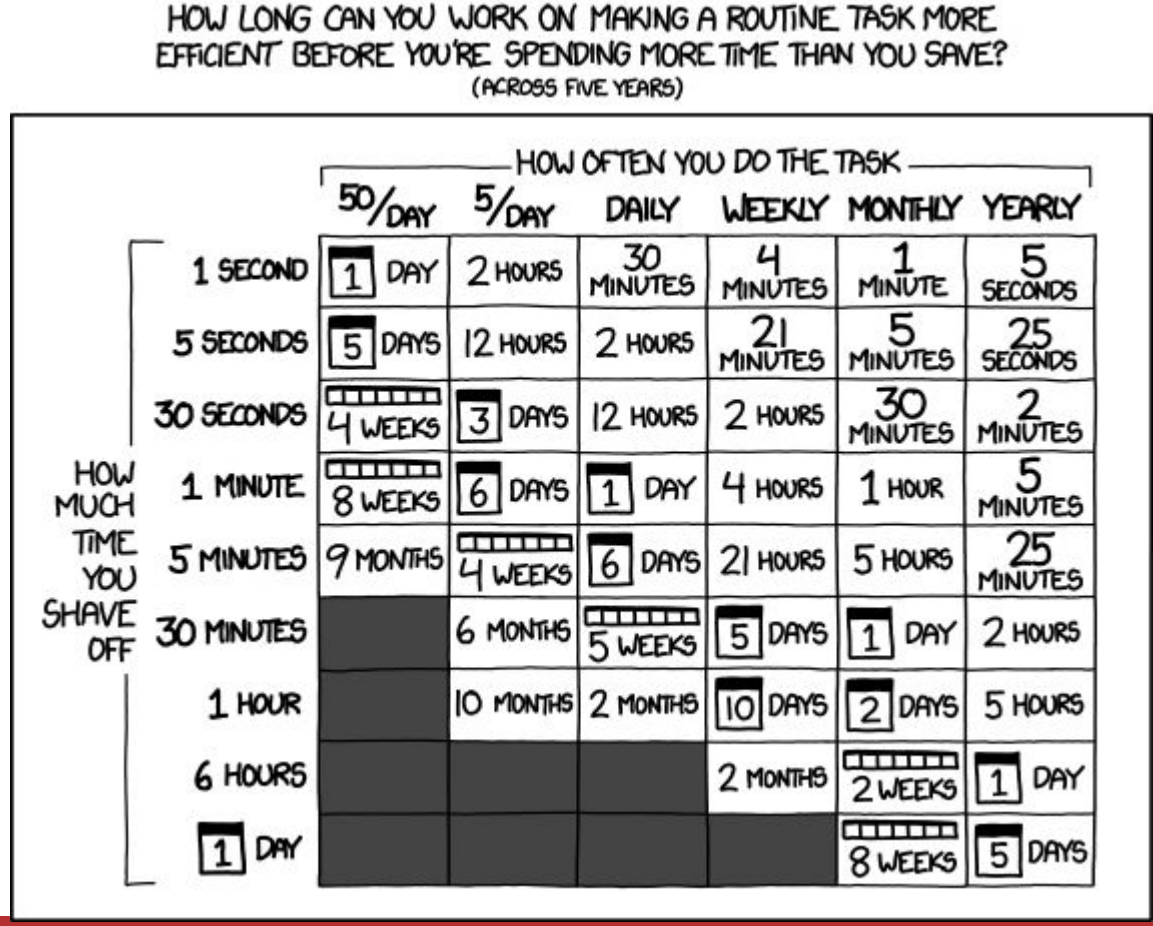
Optimisation

Premature optimisation should be avoided.

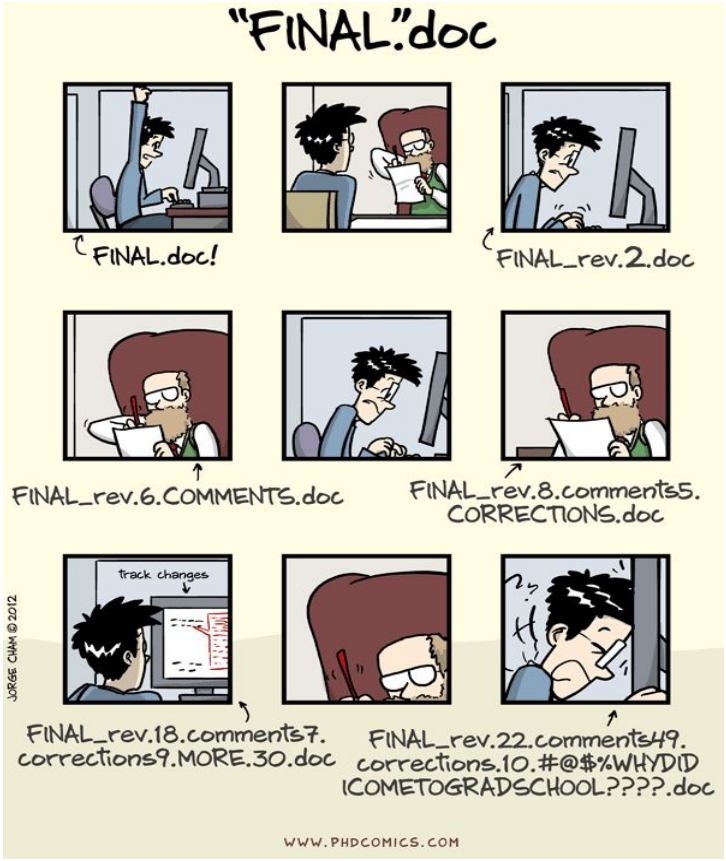
Knowing **when** to optimise is as important as knowing **how**

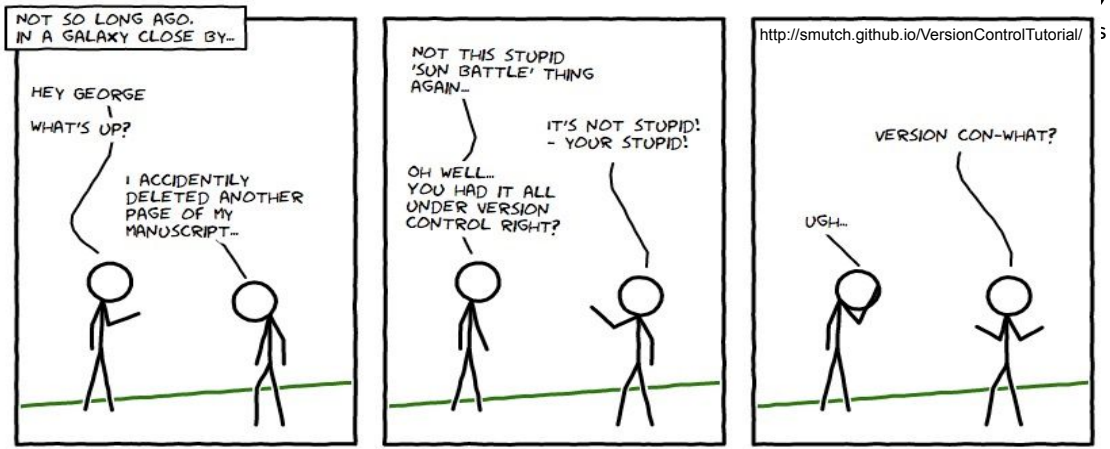
Good coding style™ can make optimisation easy

https://xkcd.com/1205/



The Problem





Version control, a.k.a. revision control / source code management, is basically a system for **recording and managing changes made to files and folders.**

You can track:

- source code (e.g. Python, R, Bash scripts),
- other files containing mostly text (e.g. LaTeX, csv, plain text),
- work by a lone developer, or
- collaboration on projects (track who's done what, branch to develop different streams, etc).

Drag your dot to how you are feeling:

I use git all the time and am a branch master

I use git occasionally.

This is literally the first time I heard about git.

Students, drag the icon!

Why Version Control?

ASTRONOMY DATA AND COMPUTING SERVICES

As researchers, we spend much of our time writing code, whether it be for data cleaning, data analysis and modelling, machine learning, or visualisation. As such, our codes are often constantly evolving. By putting all of our code under version control we can:

- **tag code** versions for later reference *(via tags)*.
- record a **unique identifier** for the exact code version used to produce a particular plot or result *(via commit identifiers)*.
- **roll back** our code to previous states *(via checkout)*.
- **identify** when/how **bugs** were introduced *(via diff/blame)*.
- **keep multiple versions** of the same code in sync with each other *(via branches/merging)*.
- efficiently **share and collaborate** on our codes with others *(via remotes/online hosting)*.

Why Version Control?

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

It's important to also realise that many of the advantages of version control are not limited to just managing code. For example, it can also be useful when writing papers/reports. Here we can use version control to:

- **bring back** that paragraph we accidentally deleted last week.
- **try out a different structure** and simply disregard it if we don't like it.
- **concurrently work on a paper** with a collaborator and then **automatically merge** all of our **changes** together.

The upshot is ***you should use version control for almost everything***. The benefits are well worth it...

Summary and cheat sheet

ADACS
ASTRONOMY DATA AND COMPUTING SERVICES

Writing good code takes practice.

Reuse things that work for you.

Develop a support group you can call on for help.
We have hacky-hour and STDERR - weekly meetup groups.

Share your codes on GitHub or similar, with documentation, so others can benefit from your work.

Oh, and publish your code and cite that of others!!!