WELCOME

PROGRAMMING WITH PYTHON

Lucy Whalley

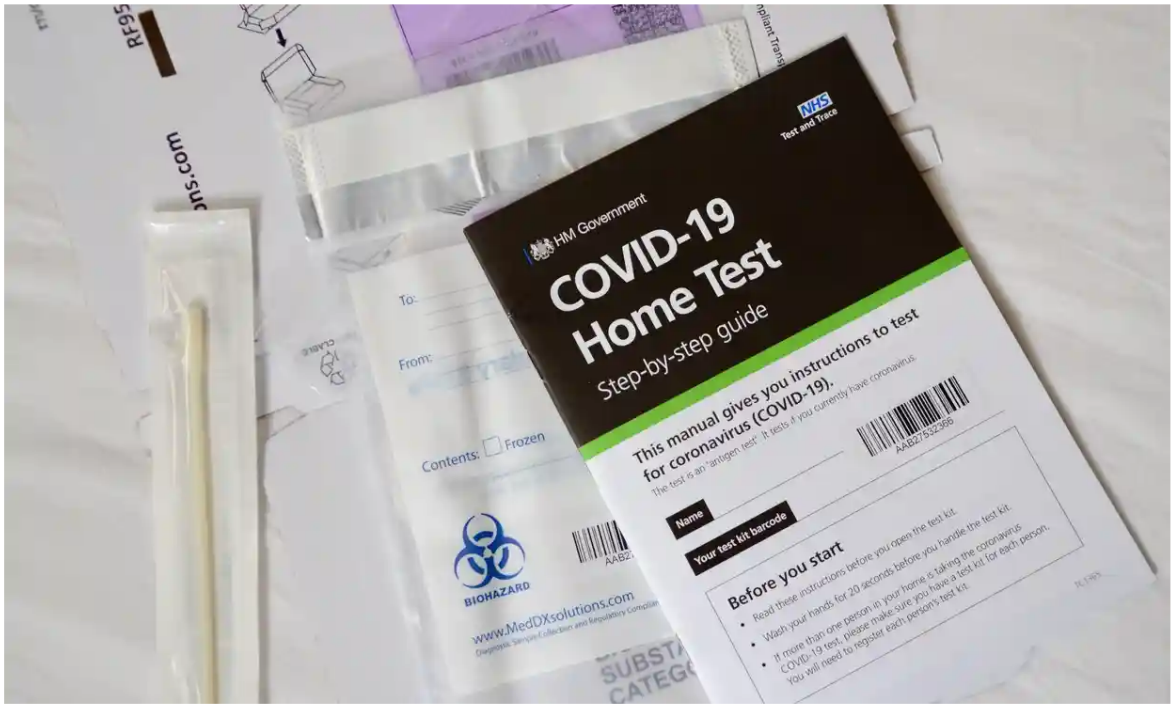
lucydot.github.io/slides

lucydot.github.io/python_novice

Covid: how Excel may have caused loss of 16,000 test results in England

Public Health England data error blamed on limitations of Microsoft spreadsheet

- Coronavirus – latest updates
- See all our coronavirus coverage



WHY PROGRAMMING? REPRODUCIBILITY

Reproducibility is a major principle of the scientific method. It means that a result obtained by an experiment or observational study should be achieved again with a high degree of agreement when the study is replicated with the same methodology by different researchers.

WHY PROGRAMMING? REPRODUCIBILITY

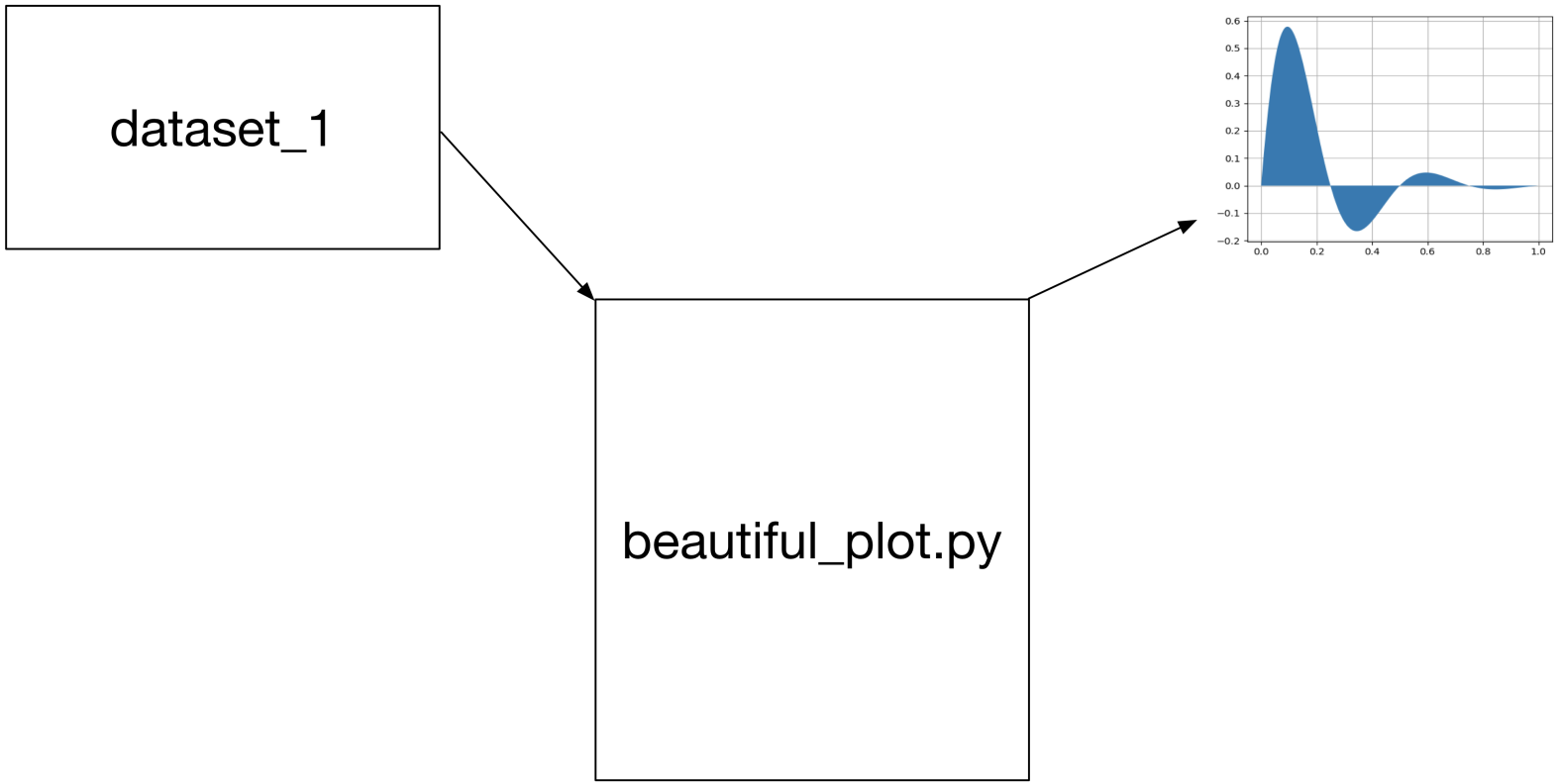
The work below relates to sections **Hot polaron states** and **Carrier cooling: heat transfer to the lattice** of the 2017 publication *Slow cooling of hot polarons in halide perovskite solar cells*. The notebook was written by Lucy D. Whalley.

```
# import libraries
import math
import scipy
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import scipy.integrate as integrate
%matplotlib inline
```

```
# constants in SI
m = 9.10938291E-31 # mass of electron
q = 1.60217657E-19 # charge of electron
ε = 8.85418782E-12 # permittivity of free space
h = 6.62606957E-34 # plancks magical number
hbar = 1.054571800E-34 # reduced planck
Rydberg = 2.1787E-18 # Rydberg energy (in joules)
avogadro = 6.02214179E23

# units in SI
Å = 1E-10 # m to angstrom
k = 1.38064852E-23
```

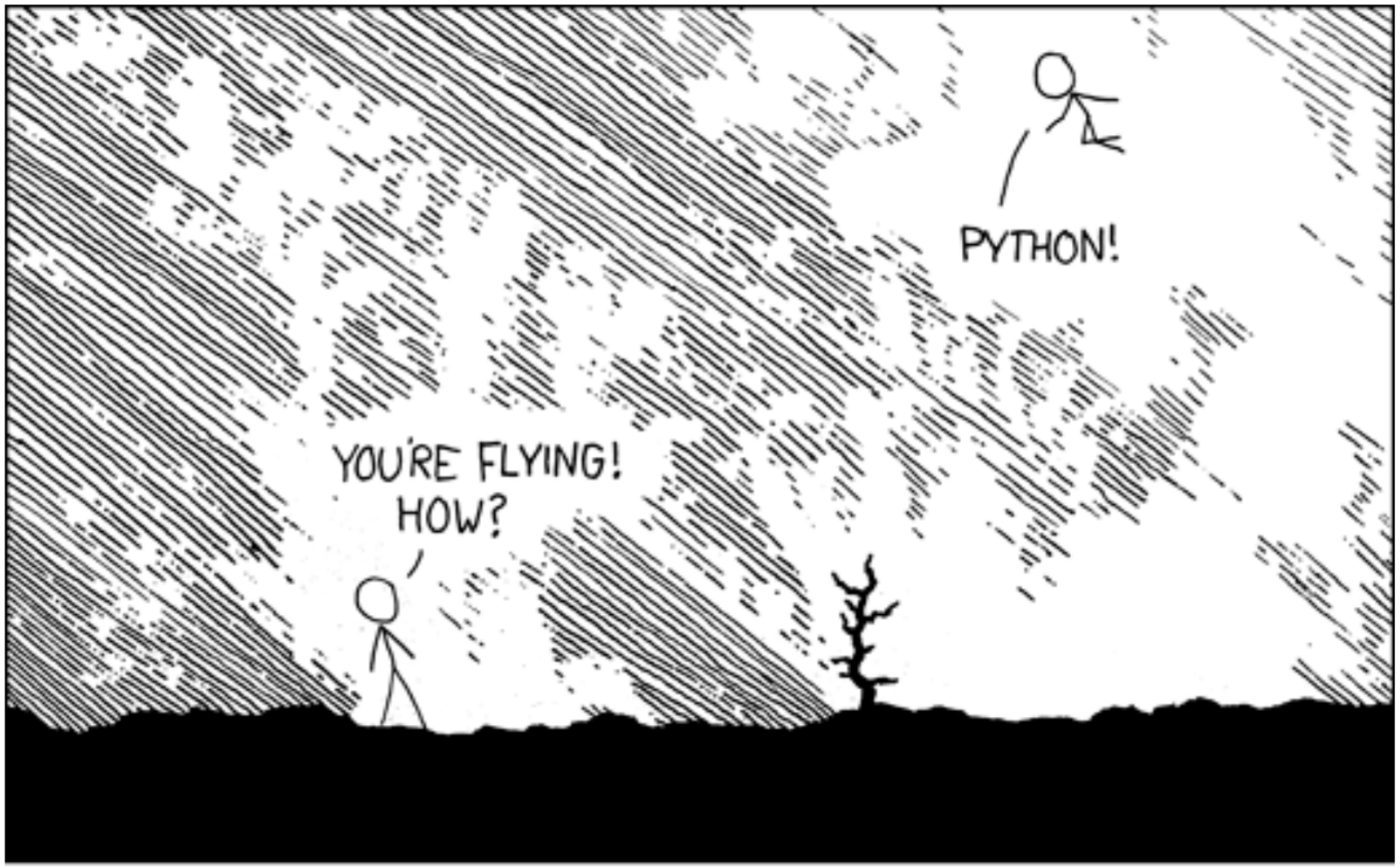
Paper: Slow Cooling of Hot Polarons in Halide Perovskite Solar Cells
Analysis code: github.com/WMD-group/hot-carrier-cooling

WHY PROGRAMMING? REPEATABILITY

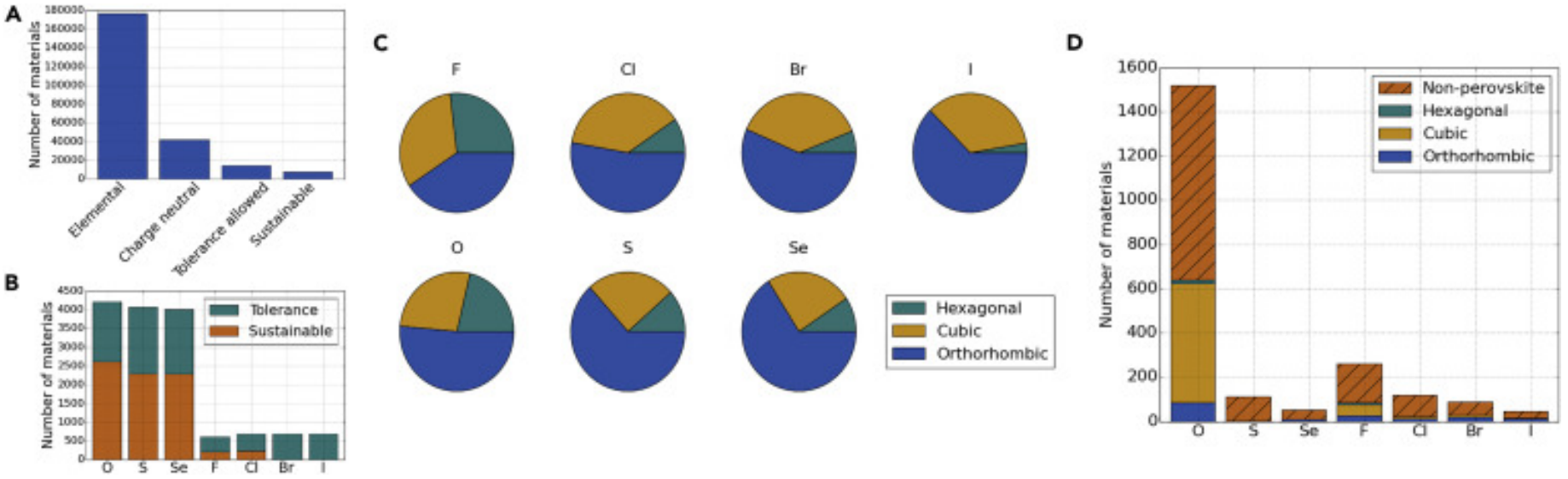


WHY PROGRAMMING? REPEATABILITY

WHY PYTHON? IT GIVES YOU WINGS

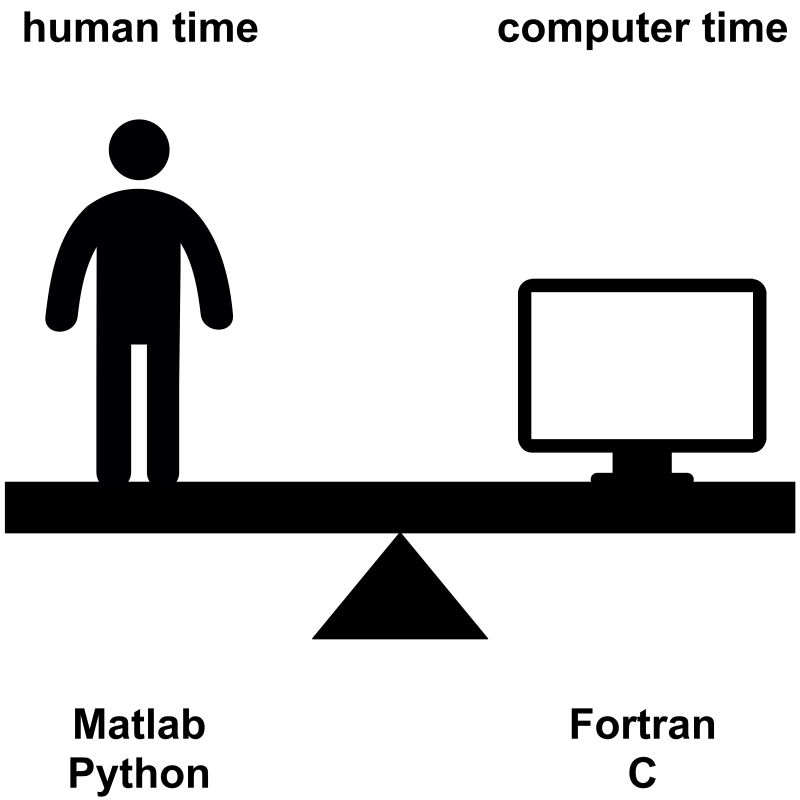


WHY PYTHON? PUBLICATION-READY PLOTS



Computational Screening of All Stoichiometric Inorganic Materials, D.W. Davies et al.

WHY PYTHON? THE TRADE-OFF



WHY PYTHON? EVERYTHING ELSE...

- readable
- free to use
- cross-platform
- well documented
- widely used

OUTLINE

1. running python code
2. variables
3. data types
4. functions, help and errors
5. lists
6. for loops
7. if statements

PLAIN TEXT VS. JUPYTER NOTEBOOK

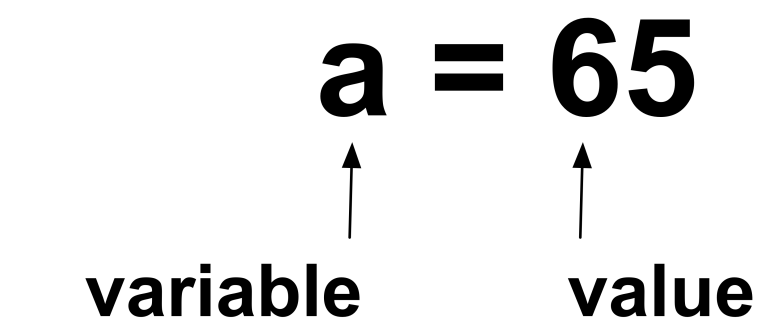
- *Jupyter notebook approach:*
  - write code in a `jupyter notebook`
  - run code in a `jupyter notebook`
  - save with a `.ipynb` extension
- *Plain text approach:*
  - write code in a text editor
  - save with a `.py` extension
  - run code using a terminal

TASK

Use your Jupyter notebook to...

- link to the CDT-ReNU webpage
- calculate 3624357/325
- make a bullet pointed ToDo list with heading "ToDo list"

[Thumbs up when you're done please]



- letters, digits and _
- cannot start with a digit
- _ start has special meaning
- case sensitive

TASK

What is the final value of `position` in the program below? (Try to predict without running the code!)

```
initial = 'left'
position = initial
initial = 'right'
```

TASK

What do you think the following code will print?

```
first = 1
second = 5*first
first=2
print('first is', first, 'and second is', second)
```

DATA TYPES

Data type	Python name	Definition	Example
integer	int	positive or negative whole numbers	`-256`
float	float	real number	`-3.16436`
string	str	character string	`"20 pence."`
list	list	a sequence of values	`['frog',2,8]`

+ boolean, dict, tuple, complex, None, set

TASK

Which of the following will print 2.0? Note there may be more than one correct answer.

```
first = 1.0
second = "1"
third = "1.1"
```

1. `first + float(second)`
2. `float(second) + float(third)`
3. `first + int(third)`
4. `first + int(float(third))`
5. `int(first) + int(float(third))`
6. `2.0 * second`

OUTLINE

1. **running python code:** Jupyter Notebooks, markdown basics
2. **variables:** variable names, variable assignment, `print()`, execution order
3. **data types:** integer, float, string, list, `len()`, string operations/indexing/slicing, type conversion: `int()`, `str()`, `float()`
4. **functions, help and errors:** `min()`, `max()`, `round()`, `help()`, runtime errors (exceptions), syntax errors
5. **lists**
6. **for loops**
7. **if statements**

LISTS

Data type	Python name	Definition	Example
integer	int	positive or negative whole numbers	`-256`
float	float	real number	`-3.16436`
string	str	character string	`"20 pence."`
list	list	a sequence of values	`['frog',2,8]`

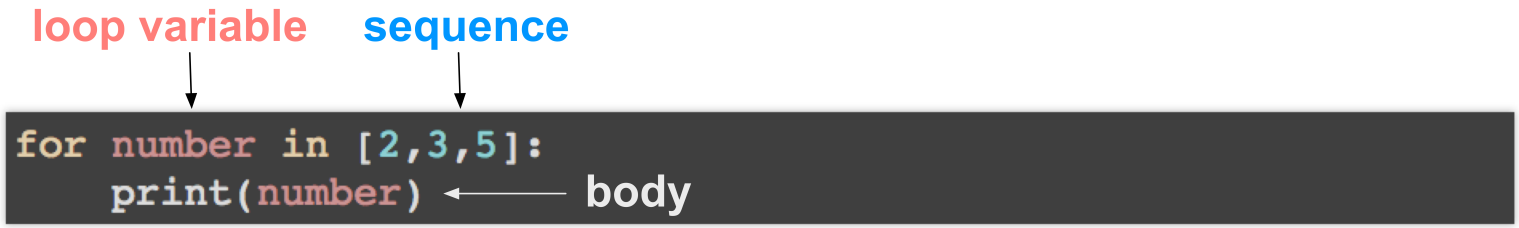
FOR LOOPS

```
print(2)
print(3)
print(5)
```

```
for number in [2,3,5]:
    print(number)
```

FOR LOOPS

```
print(2)
print(3)
print(5)
```



TASK

I want to sum the integers from 1 to 10. What is wrong with this code? How can I fix it?

```
total = 0
for number in range(10):
    total = total + number
print(total)
```

CONDITIONALS

```
mass = 4.2

if mass > 3:
    print(mass, ' is large')

if mass < 2:
    print(mass, ' is small')

if 2 <= mass <= 3:
    print(mass, ' is just right')
```

TASK

What is wrong with the code? Fix the code so that it works as intented.

```
grade = 95

if grade >= 70:
    print("grade is C")
elif grade >= 80:
    print("grade is B")
elif grade >= 90:
    print("grade is A")
```

SUMMARY

1. **running python code:** Jupyter Notebooks, markdown basics
2. **variables:** variable names, variable assignment, `print()`, execution order
3. **data types:** integer, float, string, list, `len()`, string operations/indexing/slicing, type conversion: `int()`, `str()`, `float()`
4. **functions, help and errors:** `min()`, `max()`, `round()`, `help()`, runtime errors (exceptions), syntax errors
5. **lists:** sequence type, immutable vs mutable, list method append, del
6. **for loops:** dummy variable, loop syntax, index from 0
7. **if statements:** if, elif, else, ordering

These slides available at: lucydot.github.io/slides
Workshop materials are available at: lucydot.github.io/python_novice

Back tomorrow at 10.15am for Python Part Two

OUTLINE

1. functions
2. variable scope
3. libraries
4. cleaning data with pandas
5. analysing data with numpy
6. plotting data with matplotlib
7. running code as a Python script
8. Programming good practice

FUNCTIONS

```
def print_greeting():
    print ("Hello!")
```

FUNCTIONS

```
def print_personalised_greeting(name):
    print ("Hello "+name)
```

TASK

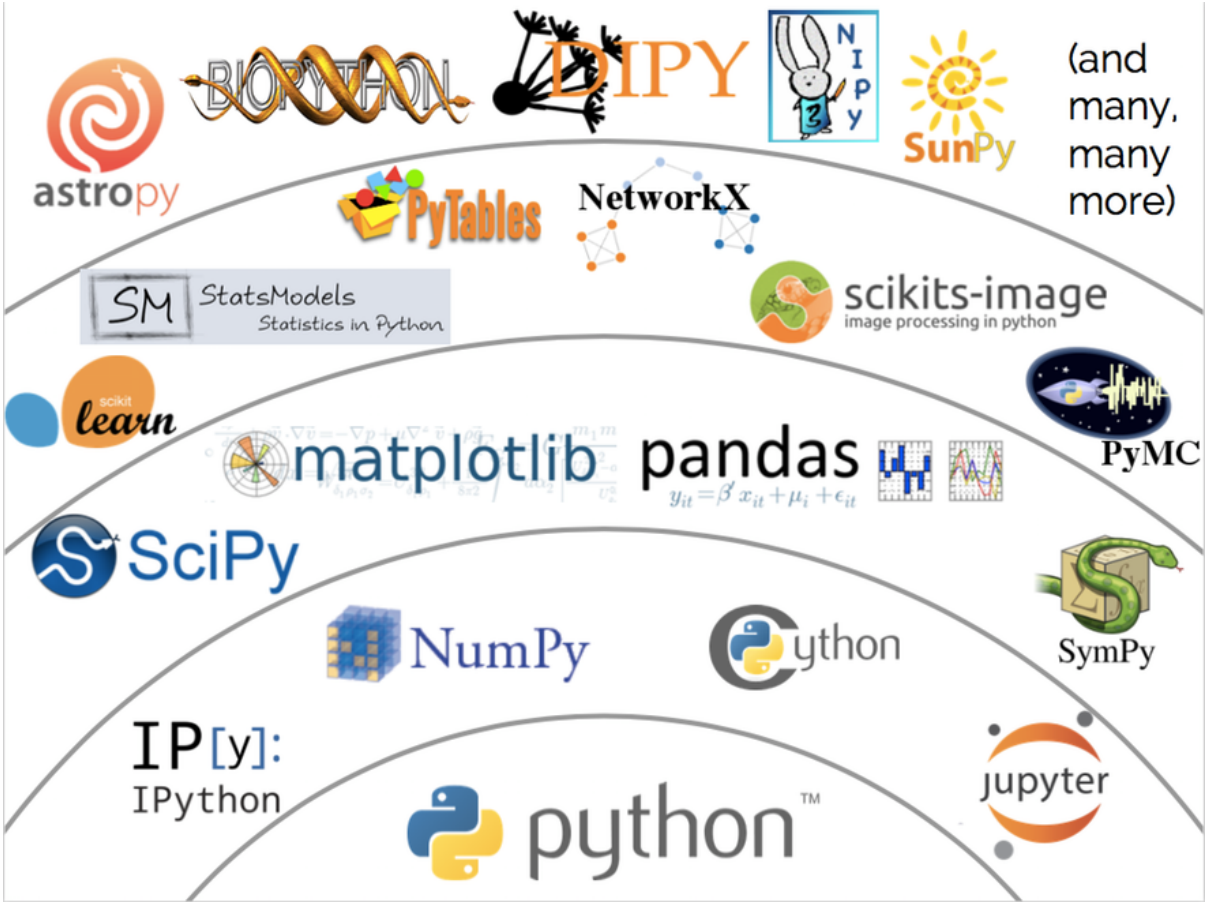
Fill in the blanks to create a function that takes a list of numbers as an argument and returns the first negative value in the list. What does your function do if the list has only positive numbers?

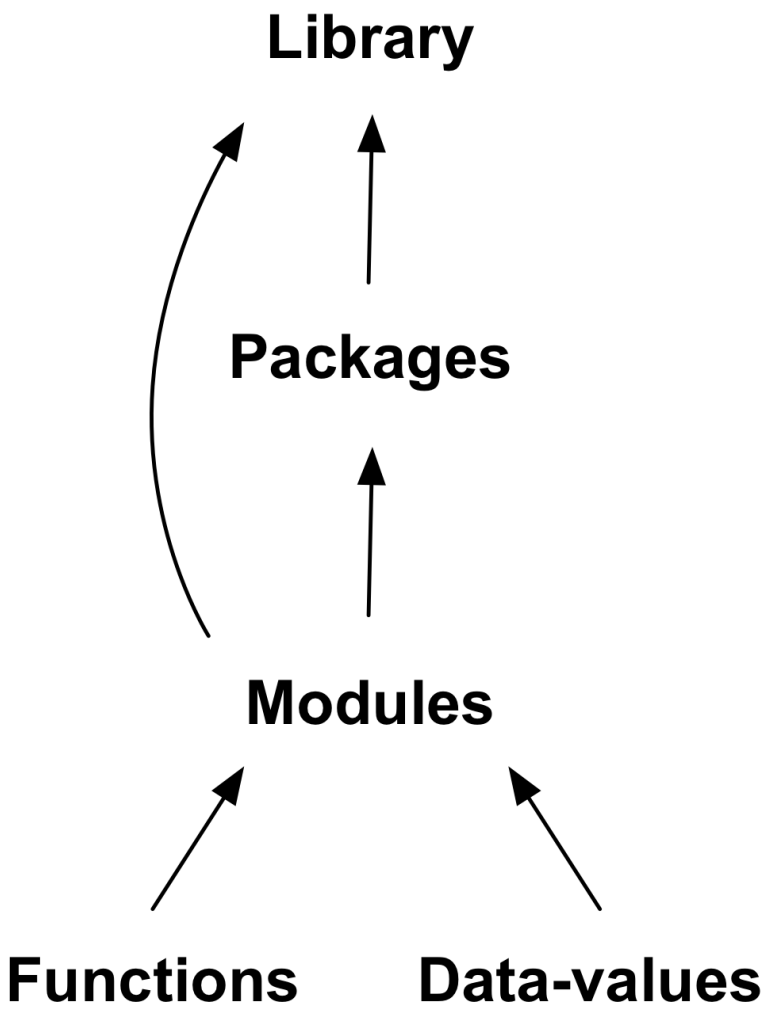
```
def first_negative(values):
    for v in ____:
        if ____:
            return ____
```

VARIABLE SCOPE

```
pressure = 103.9

def adjust(t):
    temperature = t*1.43/pressure
    return temperature
```

PYTHON SCIENTIFIC LIBRARIES




Library
Packages
Modules
Functions
Data-values

TASK

You want to select a random character from a string.
`"base = ATCHAGHRASG"`

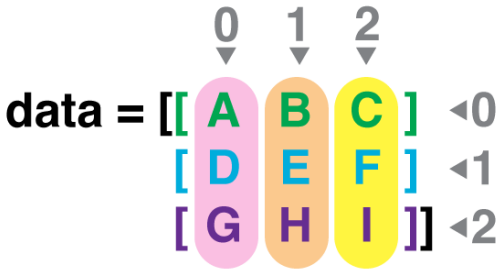
1. which standard library module could help you?
2. which function could you select from that module?
3. try to write a program that uses that function

Feel free to look online (search for "Python standard library")

OUTLINE

1. **functions**: function syntax, return statement, parameters and arguments
2. **variable scope**: local and global variables
3. **libraries**: modules, packages, libraries, import statements, aliases
4. **cleaning data with pandas**:
5. **analysing data with numpy**:
6. **plotting data with matplotlib**:
7. **running code as a Python script**:
8. **programming good practice**:

INDEXING ARRAYS



data[0][0] = A
data[0][1] = B
data[0][2] = C

data[1][0] = D
data[1][1] = E
data[1][2] = F

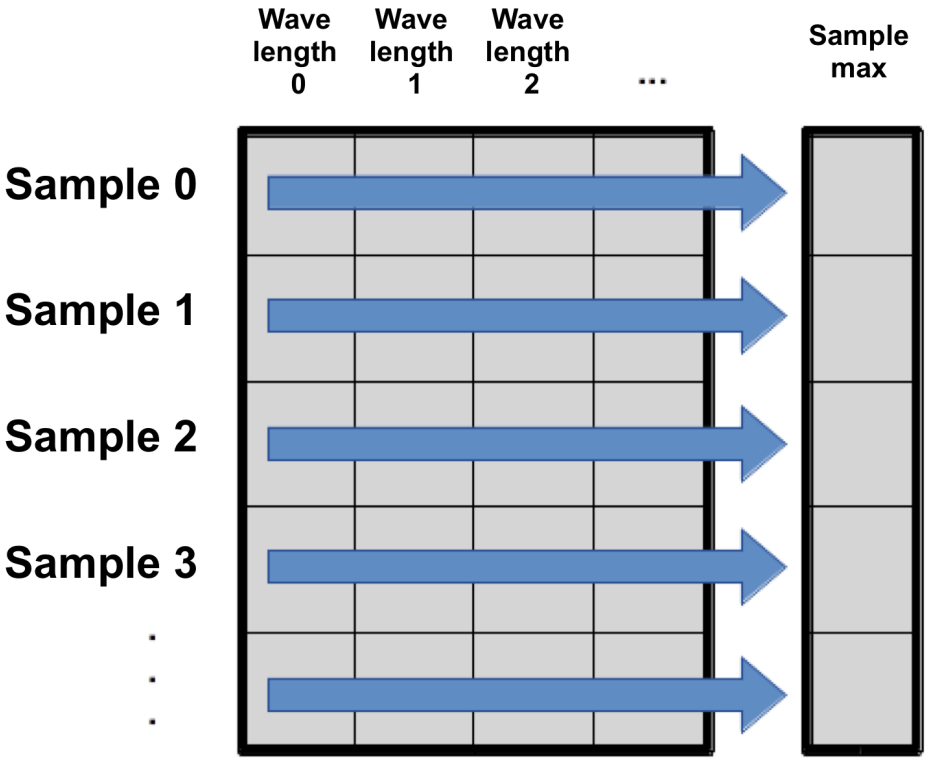
data[2][0] = G
data[2][1] = H
data[2][2] = I

TASK

Crack the code

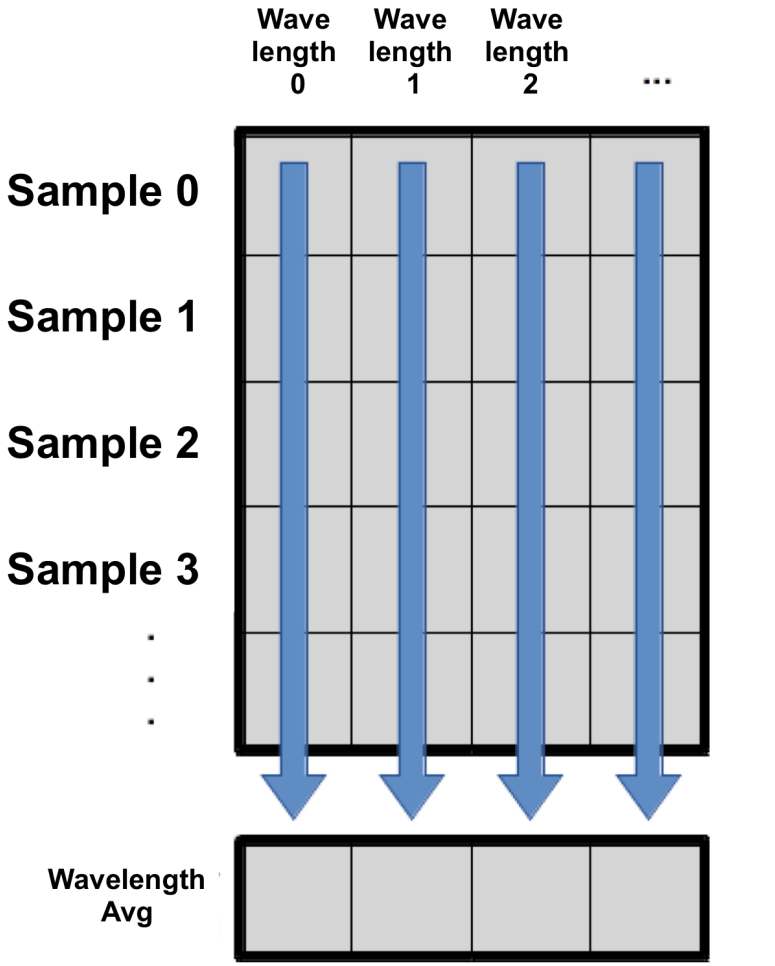
```
letters = np.array([[r,y,c,t],
                    [u,o,x,e],
                    [d,p,i,n]])

letters[0,2] letters[2,0] letters[0,3] -
letters[0,0] letters[1,3] letters[3,3] letters[0,1]
```

OPERATIONS ACROSS AXES



Max for each sample

```
data.max(axis=1)
```



Average for each wavelength

```
data.mean(axis=0)
```

PUTTING IT ALL TOGETHER

bit.ly/plot_example

PROGRAMMING GOOD PRACTICE

Follow standard Python style

- focus on readability (consistency is key!)
- maximum line length of 79 characters
- whitespace:

```
spam(ham[1], {eggs: 2})
spam( ham[ 1 ], { eggs: 2} )
```

- clear, meaningful variable names

PROGRAMMING GOOD PRACTICE

Use assertions to check for errors

```
def calc_bulk_density(mass,volume):
    assert volume > 0
    return mass / volume
```

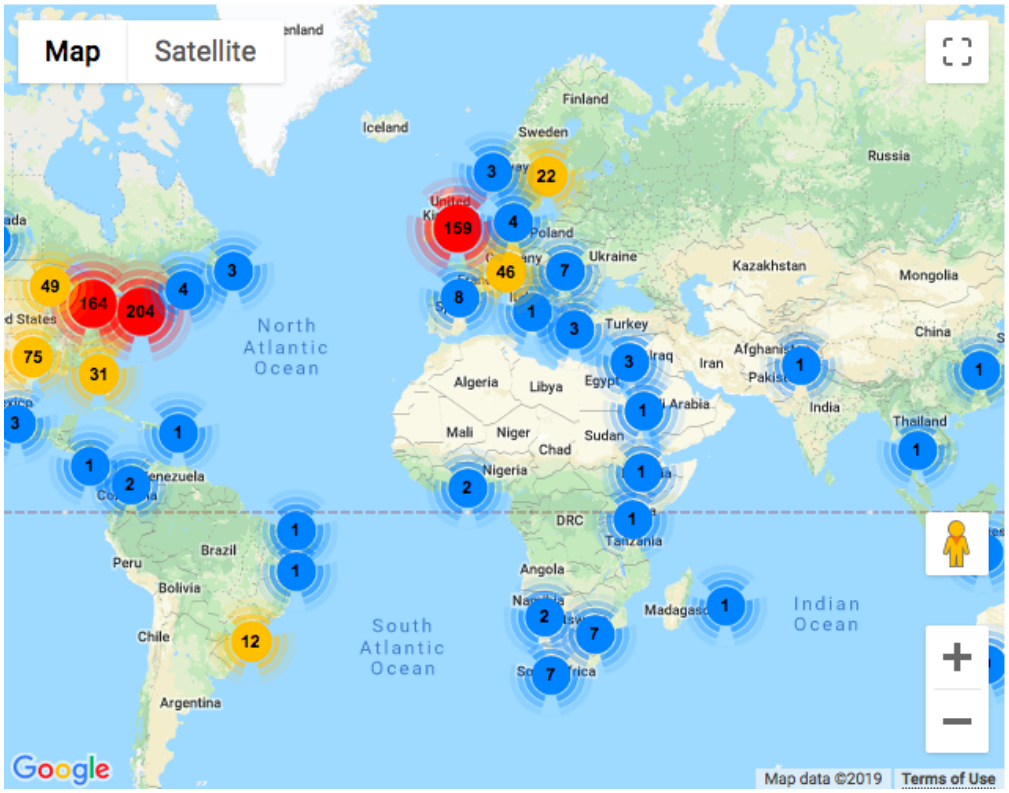
PROGRAMMING GOOD PRACTICE

Document your code with docstrings

```
def calc_bulk_density(mass,volume):
    "Return dry bulk density = powder mass / powder volume."
    assert volume > 0
    return mass / volume
```

PROGRAMMING GOOD PRACTICE

Use version control - attend a git SWC



software-carpentry.org

SCIENTIFIC GOOD PRACTICE

Aim for reproducibility

Imports and preamble

In [9]:
```
# plots displayed within notebook
%matplotlib inline

# import scientific libraries
import math
import matplotlib.pyplot as plt
import numpy as np

# import modules from the effmass package
from effmass import inputs, analysis, extrema, outputs, dos, ev_to_hartree
```

Effective mass and non-parabolicity of CdTe

Settings

First we use the `inputs` module to create a `Settings` object. The `extrema_search_depth` attribute tells us how far from the CBM/VBM we would like to search for the bandstructure minima/maxima. The `energy_range` attribute sets the energy range for each band `Segment`.

In [2]:
```
settings = inputs.Settings(extrema_search_depth=0.075, energy_range=0.25)
```

Import bandstructure data

We now use the `inputs` module to create a `Data` object which automatically imports the vasp data from the files specified. We manually specify how many k-points to ignore at the start of each file. These are the k-points which are included as part of the non-self-consistent bandstructure calculation, but which do not form part of the bandstructure itself.

paper: *Impact of nonparabolic electronic band structure on the optical and transport properties of photovoltaic materials*, L.D. Whalley et al.
code: github.com/lucydot/effmass

SUMMARY

1. **functions**: function syntax, return statement, parameters and arguments
2. **variable scope**: local and global variables
3. **libraries**: modules, packages, libraries, import statements, aliases
4. **cleaning data with pandas**: `pandas.read_csv`, DataFrames, `pandas.to_csv`
5. **analysing data with numpy**: `numpy.loadtxt`, N-dimensional arrays, attributes
6. **plotting data with matplotlib**: `%matplotlib inline`, `plot()`, `xlabel()`, `ylabel()`, `show()`, `savefig()`
7. **running code as a Python script**: `%%writefile filename.py`, `python3 filename.py`
8. **programming good practice**: Python style, `assert` statements, docstring

1. **running python code:** Jupyter Notebooks, markdown basics
2. **variables:** variable names, variable assignment, `print()`, execution order
3. **data types:** integer, float, string, list, `len()`, string operations/indexing/slicing, type conversion: `int()`, `str()`, `float()`
4. **functions, help and errors:** `min()`, `max()`, `round()`, `help()`, runtime errors (exceptions), syntax errors
5. **lists:** sequence type, immutable vs mutable, list method append, del
6. **for loops:** dummy variable, loop syntax, index from 0
7. **if statements:** if, elif, else, ordering
8. **functions**: function syntax, return statement, parameters and arguments
9. **variable scope**: local and global variables
10. **libraries**: modules, packages, libraries, import statements, aliases
11. **cleaning data with pandas**: `pandas.read_csv`, DataFrames, `pandas.to_csv`
12. **analysing data with numpy**: `numpy.loadtxt`, N-dimensional arrays, attributes
13. **plotting data with matplotlib**: `%matplotlib inline`, `plot()`, `xlabel()`, `ylabel()`, `show()`, `savefig()`
14. **running code as a Python script**: `%%writefile filename.py`, `python3 filename.py`
15. **programming good practice**: Python style, `assert` statements, docstring

Well done!...keep going...

====*Thank-you*====