Python Basics

What to expect

- Understand basic python
  - Write your own scripts for research/fun
  - Understand other people's scripts
- Perform simple image analysis in python
  - Learn about useful libraries
  - Do some basic image processing tasks
- Get an idea of the power of python
- Thinking like a computer
  - Most important part of programming!

What not to expect

- Achieve Zen mastery of python
  - Not possible in 8 hours
- “Professional quality” code
  - Requires years of practice
  - Doesn’t really mean anything anyway
- Code without mistakes
  - Everyone makes mistakes
  - Sometimes they are incredibly subtle

The Plan

- Why Python?
- Python: getting started.
- Python: Introduction to the language.
- Programming:
  - Data types (and some python specific ones)
  - Variables
  - Operators
  - Program control
  - Functions
- Some examples
- Python data analysis: Plotting mathematical functions

Why Python?

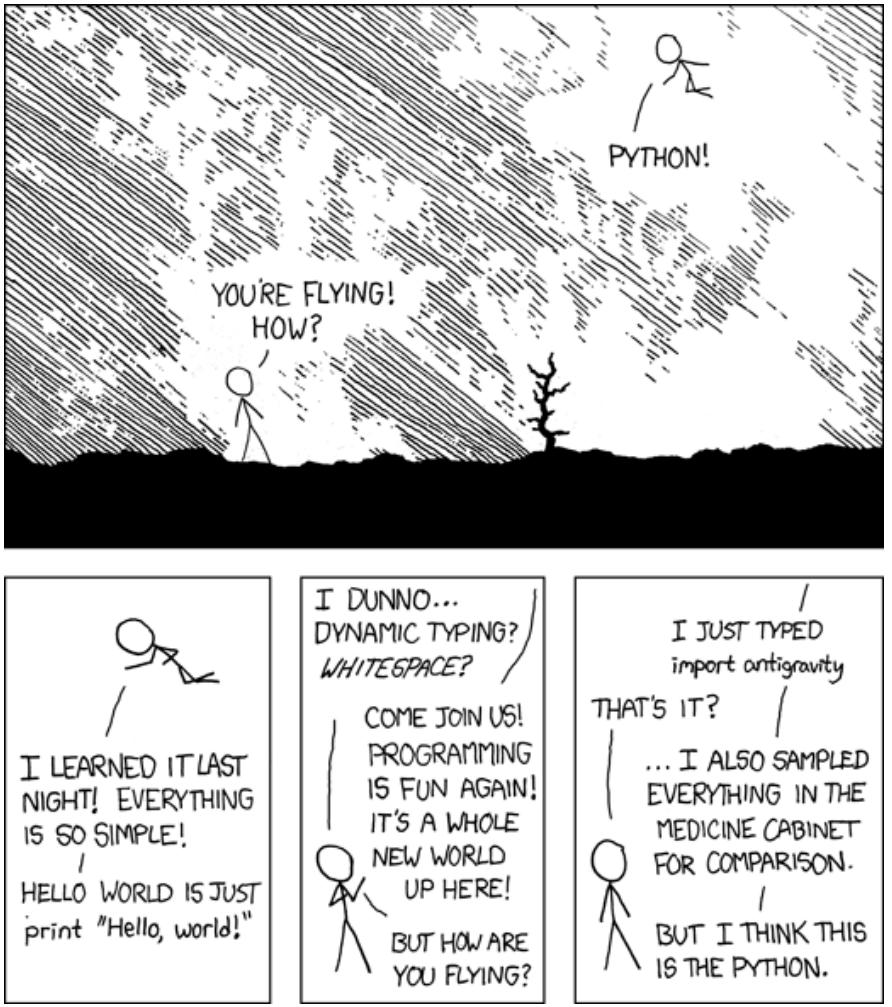
- There are lots of programming languages
  - C/C++
  - R
  - Brainf**k
- Python is relatively simple, but still very powerful.
- It's a 'gateway' language.

```
draw_dendrogram = function(hc, gaps, horizontal = T){
    h = hc$height / max(hc$height) / 1.05
    m = hc$merge
    o = hc$order
    n = length(o)

    m[m > 0] = n + m[m > 0]
    m[m < 0] = abs(m[m < 0])

    dist = matrix(0, nrow = 2 * n - 1, ncol = 2, dimnames = list(NULL, c("x", "y")))
    dist[1:n, 1] = 1 / n / 2 + (1 / n) * (match(1:n, o) - 1)

    for(i in 1:nrow(m)){
        dist[n + i, 1] = (dist[m[i, 1], 1] + dist[m[i, 2], 1]) / 2
        dist[n + i, 2] = h[i]
    }

    draw_connection = function(x1, x2, y1, y2, y){
        res = list(
            x = c(x1, x1, x2, x2),
            y = c(y1, y, y, y2)
        )

        return(res)
    }

    x = rep(NA, nrow(m) * 4)
    y = rep(NA, nrow(m) * 4)
    id = rep(1:nrow(m), rep(4, nrow(m)))

    for(i in 1:nrow(m)){
        c = draw_connection(dist[m[i, 1], 1], dist[m[i, 2], 1], dist[m[i, 1], 2], dist[m[i, 2], 2], h[i])
        k = (i - 1) * 4 + 1
        x[k : (k + 3)] = c$x
        y[k : (k + 3)] = c$y
    }
```

```
--<-<<+[+[<+>--->->->-<<<]>]<<--
.<++++++.<<-..<<.<+.>>.>>.<<<.+++.>>.>>-
.<<<+.
```

Python: Getting started

- Python is a high-level language
  - Means it takes care of the tedious stuff for you
- It is also dynamically typed
  - You don't have to explicitly say what type of data you are using.
- Python can be run one line at a time in a command prompt
  - Useful for trying things out



Programming

- Programs consist of a few basic elements:
  - Variables
  - Program control statements
  - Functions
- Writing a program is an incremental process
  - Get smaller bits to work on their own
  - Good code is structured to do this implicitly
    - Functions

Variables

- Variables are where we store data used in the program.
  - Kind of like a labeled box
- Variables have a name, a type and a value.
  - **In python, a variable's type is decided when you run the program.**
  - Variable names are case sensitive, and must not start with a number or symbol (except _)
  - There are also some reserved names that you can't use:

```
and assert break class continue def
del elif else except exec finally
for from global if import in is
lambda not or pass print raise
return try while
```

Programming: Basic data types

- There are five basic data types:
  - Integers
    - E.g. 10
  - Floating point numbers
    - E.g. 3.14
  - Strings
    - E.g. “Hello, World!”
  - Booleans
    - True/False
  - None
    - Literally nothing
- In python, you don’t need to say which one you’re using – the interpreter will decide.
  - This can cause some problems (see later when we talk about operators)
  - If you need to know what type something is, use the builtin function type() on it. E.g. type(10) → int, type(3.14) → float.

Strings

```
>>> a = “Spam, eggs and spam.”
>>> a.startswith(“S”)
True
>>> a.endswith(“m”)
False
>>> a.split()
[“Spam,”, “eggs”, “and”,
“spam.”]
>>> a.split(“,”)
[“Spam”, “ eggs and spam.”]
>>> a.strip(“.”)
Spam, eggs and spam
>>> s = ['spam', 'spam',
'spam', 'spam', 'spam', 'spam',
'spam', 'spam', 'spam', 'spam']
>>> “, ”.join(s)
'spam, spam, spam, spam, spam,
spam, spam, spam, spam, spam'
```

- The string type in python has ‘methods’
- Some that I use a lot are
  - startswith()
  - endswith()
  - split()
  - strip()
- Also useful is the join() method

Python specific types: Tuples

- Tuples are a data structure
  - Just means they contain more than one thing
- Tuples are immutable
  - Once created, can't be changed
- Useful for:
  - Vectors (x,y,z)
  - Function returns (see later)
  - Storing things you need to keep unchanged
- Get things out of them using the square brackets operator
  - Index starts at zero in python.
  - More on operators later…

```
>>> a = tuple(1,2,3)
>>> b = (1,2,3)
>>> c = (1, "two", 3.0)
>>> d = (1, ("This", "is", "fine"), 2)

>>> d[1]
("This", "is", "fine")
```

Python specific types: Lists

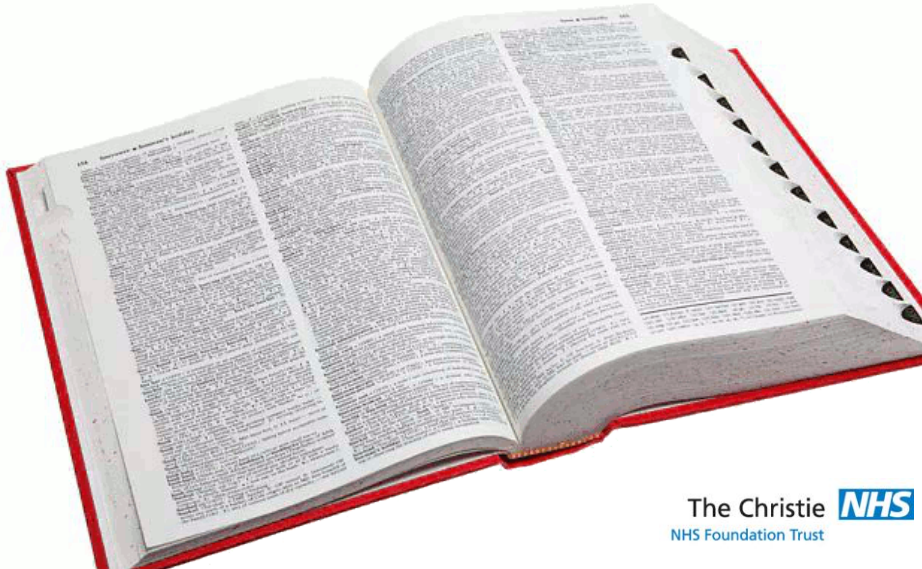
```
>>> a = list(1, 2, 3)
>>> b = [1,2,3]
>>> c = b.append(4)
>>> c
[1,2,3,4]
>>> d = c.extend([5,6,7])
>>> d
[1,2,3,4,5,6,7]
```

- Very similar to tuples
  - In this case, they can be changed
- There are a few methods in the list data structure
  - Basically, things you can do to/with them
  - Most important are append and extend
  - To see them all, do dir(list)
- Useful for:
  - Lists of files
  - Images
  - Iterating over (see later)

Python specific types: Dictionaries

- Dictionaries are a key-value map
  - You give them a key, you get a value
- The keys and values can be anything you like
  - Even lists, tuples or other dictionaries!
- Dictionaries can be extended
  - Just assign your new entry
  - Trying to access one that isn't there causes an error
- Useful for:
  - Patient ID – image mapping
  - Program settings

```
>>> a = {"one":1, "two":2}
>>> a["three"] = 3
>>> a["three"]
3
>>> a["four"]
KeyError: 'four'
```



Operators

- Operators are what you use to actually do stuff
  - There are loads of them
- The obvious ones include:
  - Mathematical operators: `+ - / *`
  - Assignment operator: `=`
  - Index operator: `[ ]`
- But there are also:
  - Compound operators: `+= -= *= /=`
  - Exponentiation, modulo and floor division: `** % //`
  - Conditional operators: `== != >= <= > <`
  - Binary operators: `& | ^ ~ << >>`
  - Logical operators: `in, not in`
  - Identity operators: `is, is not`

Operator examples

```
>>> 1 + 2
3
>>> 1.0 + 2 # Note - float + int
3.0
>>> 2 * 3.0
6.0
>>> 1/3 # Note - integer division
0
>>> 1/3.0
0.33333333333
>>> 2**10 -- exponentiation
1024
>>> 3 % 2 # modulo division - 3/2 = 1 remainder 1
1
>>> 4.0 // 3.0 # Should be 1.33333, but floor rounds to 1.0
1.0
>>> # Brackets work like in normal maths
>>> (1.0 + 3.0) ** 5
1024.0
```

A word on versions

- There are two widely used versions of python: python 2 and python 3
- They behave differently!
  - Can be in subtle ways
  - E.g. in python 2, 1/3 = 0 whereas in python 3 1/3 = 0.333333
- This course teaches python 3
  - Most stuff is directly equivalent
- Python 2 is dead
  - Long live python 2!

Program Control

- Program control is what you use to structure your program
- All programs consist of loops and if statements
- In python there is only one type of if statement and two types of loop.
  - They look like those shown opposite

if *condition 1*:
    code if condition1 true
elif *condition 2*:
    code if condition 2 true
else:
    code if neither true

while *condition*:
    loop here while condition true

for *variable* **in** *iterable*:
    loop code here

If-elif-else example

Consider the following code:

```
a = 81
if a % 2 == 0:
    print("a % 2 is zero")
elif a % 3 == 0:
    print("a % 3 is zero")
else:
    print("Neither case")
```

What will be printed?

While loop example

In this code:

```
t = 0
while t < 10:
    t += 3

print(t)
```

What is the value of t printed at the end?

For loop example – using range

This is a common way to use a for loop:

```
v = 0
for i in range(0, 10):
    v += i
```

What is the value of v?

For loop example – looping over list

You can use the for loop to work on anything that can be iterated.

```
s = [2,4,6,8,10]
t = []
for u in s:
    t.append(u*u)
```

What is the value of `t[2]`?

Python: Indentation

```
def GenerateSettings():
    """
    Here we read the current settings JSON, and ask the user to verify all of it, then test the client_secrets file by attempting
    if this fails, we instruct the user to get their client secrets file and re-run.
    """
    firstTime = False
    try:
        settings = json.loads(open(SETTINGS_FILE, 'r').read())
    except:
        print("Error: settings.json not found. Generating new from scratch...")
        settings = {}
        settings["compute"] = {}
        settings["oauth_storage"] = "oauth2.dat" # A default value
        firstTime = True

    # Project ID
    if "project" in settings:
        print("The project ID in the current settings file is:\n\n%s\n\nIs this correct?"%(settings["project"]))
        choice = raw_input("y/n?  ")
        if(choice.upper() == "Y"):
            pass
        else:
            newproj = raw_input("Input the correct project ID:\t")
            settings["project"] = newproj
    else:
        newproj = raw_input("Input the your project ID:\t")
        settings["project"] = newproj

    # Storage Bucket
    if "bucket" in settings:
        print("The bucket name in the current settings file is:\n\n%s\n\nIs this correct?"%(settings["bucket"]))
        choice = raw_input("y/n?  ")
        if(choice.upper() == "Y"):
            pass
        else:
            newBucket = raw_input("Input the correct bucket name:\t")
```

- In python, indentation of your code is required
  - Sometimes the IDE will do it for you.
- Indentation determines the ‘scope’ of things
  - I’ll explain scope in a minute or two…

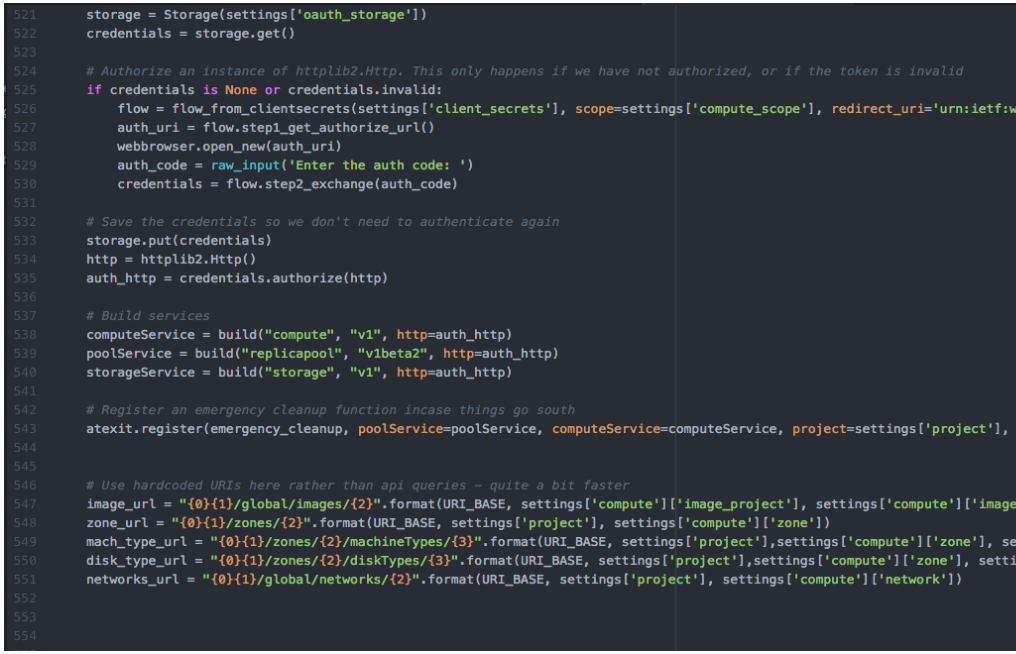
Scope

- The scope of a variable is the region of the program from which it is accessible.
  - Often you can work out the scope by looking at the indentation.
  - In this example, user_input is only available in the if statement, and only when some_condition is True.
  - Variables in a larger scope are accessible in blocks within that scope, like message in this example
- When a variable goes out of scope, it is deleted.

```
# global scope
some_condition = True
message = "SPAM"
if some_condition:
    # scope when some_condition is True
    user_input = get_input()

    if user_input == 1:
        message = "EGGS"
    else:
        message = "SPAM and EGGS"
else:
    # scope when some_condition is False
    message = "SPAM and SPAM"
# back to global scope
print(message)
```

Python: Comments

- It is very helpful if you put comments in your code
  - 6-months-from-now you won't remember what that function does.
- However, don't go mad.
- Comments in python all start '#'



```
    storage = Storage(settings['oauth_storage'])
    credentials = storage.get()

    # Authorize an instance of httplib2.Http. This only happens if we have not authorized, or if the token is invalid
    if credentials is None or credentials.invalid:
        flow = flow_from_clientsecrets(settings['client_secrets'], scope=settings['compute_scope'], redirect_uri='urn:ietf:w
        auth_uri = flow.step1_get_authorize_url()
        webbrowser.open_new(auth_uri)
        auth_code = raw_input('Enter the auth code: ')
        credentials = flow.step2_exchange(auth_code)

    # Save the credentials so we don't need to authenticate again
    storage.put(credentials)
    http = httplib2.Http()
    auth_http = credentials.authorize(http)

    # Build services
    computeService = build("compute", "v1", http=auth_http)
    poolService = build("replicapool", "v1beta2", http=auth_http)
    storageService = build("storage", "v1", http=auth_http)

    # Register an emergency cleanup function incase things go south
    atexit.register(emergency_cleanup, poolService=poolService, computeService=computeService, project=settings['project'],


    # Use hardcoded URIs here rather than api queries - quite a bit faster
    image_url = "{0}{1}/global/images/{2}".format(URI_BASE, settings['compute']['image_project'], settings['compute']['image
    zone_url = "{0}{1}/zones/{2}".format(URI_BASE, settings['project'], settings['compute']['zone'])
    mach_type_url = "{0}{1}/zones/{2}/machineTypes/{3}".format(URI_BASE, settings['project'],settings['compute']['zone'], se
    disk_type_url = "{0}{1}/zones/{2}/diskTypes/{3}".format(URI_BASE, settings['project'],settings['compute']['zone'], setti
    networks_url = "{0}{1}/global/networks/{2}".format(URI_BASE, settings['project'], settings['compute']['network'])
```

Functions

- We've been using some functions already
  - E.g. range(), print()…
- A function wraps up a piece of code so it can be used repeatedly
  - Often with slightly different parameters
- To write good code, you will need to define your own
  - Very simple to do

def *functionName***(***arguments***):**

function definition

Function example

```
def readSerial(ser):
    # intValue = st.unpack(5*'c', ser.read(4))
    intValue = int(ser.readline().strip())
    return float(intValue)/255.0

N = 512
t = np.arange(N)
data = np.zeros((N,)) + 0.1
```

```
def sayHelloTo(thing):
    print("Hello" + thing "!")

```

```
        data[i] = readSerial(ser)
        i += 1
        ii = [a % N for a in range(i+1, i+50)]
        data[ii] = 0.0
        i %= N

        aPlot.set_ydata(data)
        fig.canvas.draw()
        plt.pause(0.01)
```

Python: Libraries

```
"""
My attempt to implement the Bragg Peak equation found in Bortfeld '97, Med Phys 24 (12) 2024-

Version 0.0.1a: Working implementation, by default we use the 'inaccurate' version of the dose c

"""
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as sp

toGray = 1.602E-10

def D(R0, phi0, epsilon, sig, z):
    """
    This is the very specialised equation for water only. It's equation 28/29 in the paper
    """
    if z < (R0 - 10*sig):
        fac = (phi0)/(1.0 + 0.012*R0)
        term1 = 17.93*((R0 - z)**-0.435)
        term2 = ((0.444 + 31.7*epsilon)/R0)*((R0-z)**0.565)
        return fac*(term1 + term2) * toGray
```

- The true power of python lies in the huge number of libraries it has.
  - Pretty much a library to do anything you can think of.
  - Usually installed using a tool called pip.
- Using libraries is as simple as doing “import”
  - And knowing how to use the library, of course.

Installing libraries

- There are 200,000 ish libraries
- Searching them is really easy!
  - pip search *libraryName*
- Installing them is really easy!
  - pip install *libraryName*
- Sometimes, you might have permission errors (i.e. your user isn't allowed to install packages)
  - Try: python –m pip install *libraryName*

Tips and Tricks

- If you don't have internet access, and want to know what functions are available in a module:
  - `>>> dir(moduleName)`
  - `>>> moduleName.function.__doc__`
- There are things called 'comprehensions' which can replace some loops.
  - They build a list/dictionary from another iterable thing
  - `listName = [ fcn(a) for a in iterable ]`
  - Handy for doing some transformation on a list to get another list

Example 1: Why we are using python

```
1   #include <iostream>
```

```
1   print("Hello, World!")
```

Example 2: Find all prime numbers below 100

```
currentNumber = 2
listOfPrimes = [2]# Need to start with 2 because *everything* is divisible by 1
while currentNumber < 100:# Only looking for numbers less than 100
    isPrime = False
    for p in listOfPrimes:
        if currentNumber % p == 0:# See if the current number is divisible by any of our known primes
            isPrime = False # If it is, it can't be prime!
            break# No point checking the rest of the list
        else:
            isPrime = True # It might be prime - need to check the other numbers in our list
            continue # Jump to the next iteration

    if isPrime:
        listOfPrimes.append(currentNumber)# Add the latest number to the primes list, if it is prime.
    currentNumber += 1 # Increment the current number

print(listOfPrimes)
```

Example 3: Debug this code

```
1   def nthRoot(number, n):
2       # To take the nth root, we just
3       # exponentiate with power n^-1
4
5       if n == 0:
6           print("Impossible!")
7       return
8
9       pow = 1/n
10      return number ** pow
11
12  9throot = nthRoot(9, 9)
13  print(pow)
14
```

Brief intro to numpy

- Numpy is probably the most widely used library in python
- Much faster for numerical computations than standard python
  - Almost a factor of 10
- Handles N-dimensional arrays easily
  - Perfect for images

```
import numpy as np

N = 1024
# N values linearly spaced between two endpoints:
linear = np.linspace(start, stop, N)

# N values linearly spaced on log-scale:
log_linear = np.logspace(start, stop, N)

# 128x128x64 image:
image = np.zeros((128,128,64))

# Most maths is available in numpy
signal = np.sin(linear)

# numpy arrays understand some operators
signal_squared = signal**2
```

Brief intro to numpy

- Numpy is huge
  - No hope of even scratching the surface
  - Tomorrow's lecture will cover more
- What you will need for today:
  - Creating arrays
  - Loading data
  - Convolution

Numpy arrays

There are a few ways to create an array:

```
# Directly using a function:
image = np.zeros((128,128,64))
linear = np.linspace(0, np.pi, 1024)

# from a list
arrayList = np.array([fcn(a) for a in iterable]) # don't
do this!

# Automagically from a function
nowAnArray = np.sin(range(0, 100))
```

Loading data

Numpy has several ways to load data

```
# If you have ASCII data
txtData = np.loadtxt("fileName.txt")

# If you saved an array from a previous python run
npyData = np.load("somePreviousData.npy")

# If you have ASCII data with some bits missing
reconTXTData = np.genfromtxt("someFile.txt", …)
# This is a very very powerful function!
```

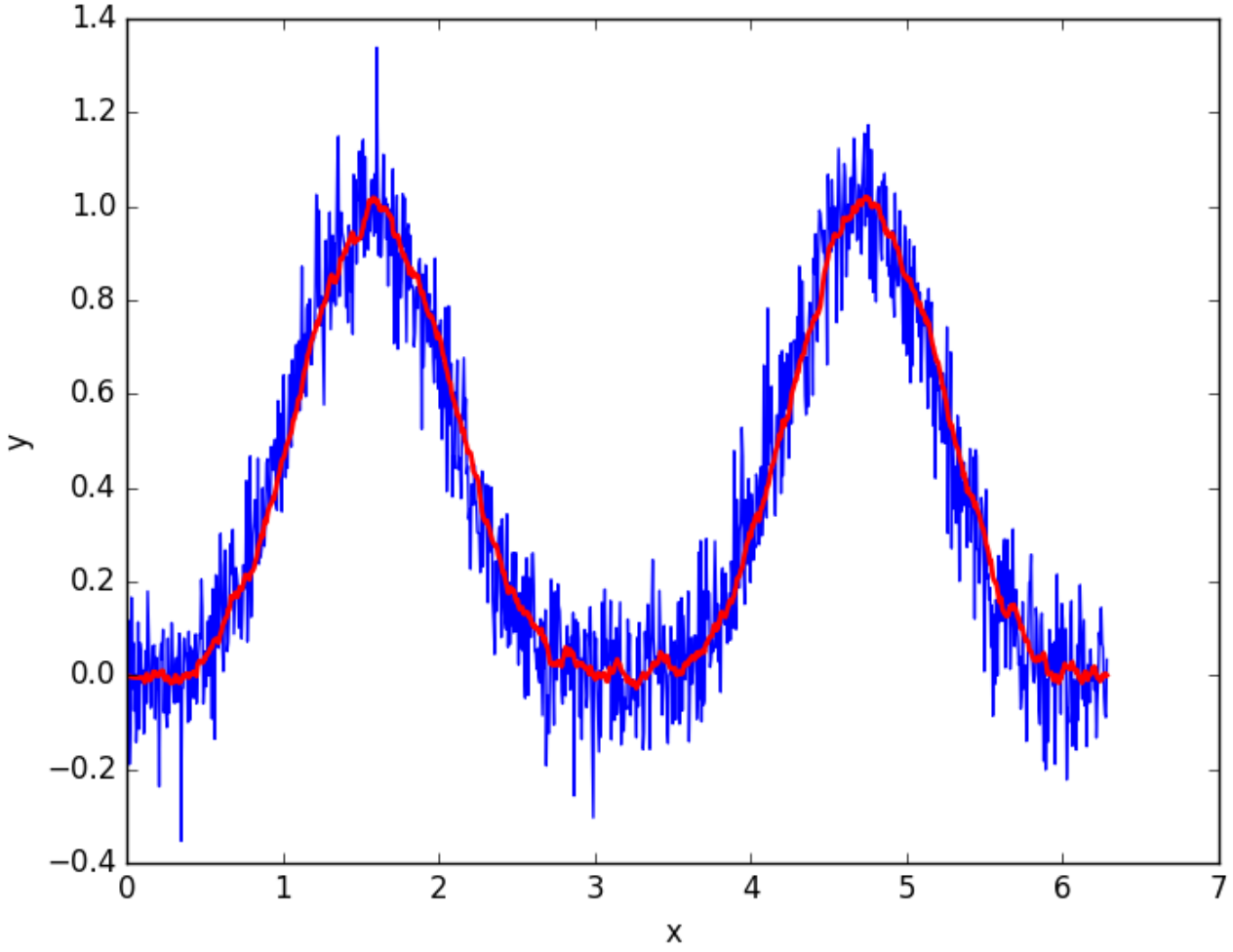
Python data analysis: plotting

- The code in the previous slide does a filtering operation on some data.
  - At the moment, we don't really know if this is giving a sensible answer or not.
- One way to see would be to plot the data
- We can use the matplotlib library to do simple plotting in python.

Code: Plotting some data with matplotlib

```python
import numpy
import matplotlib.pyplot as plt

# Create a 1024 length array of numbers between 0 and 2 pi
N = 1024
xValues = numpy.linspace(0, 2.0*numpy.pi, N, endpoint=True)

# make a noisy signal - sin^4 plus random gaussian noise
sinWithNoise = numpy.sin(xValues)**4 + numpy.random.normal(loc=0.0, scale=0.1, size=xValues.shape[0])

# Hamming window - used to do the filtering
alpha = 0.54
beta = 1.0 - alpha
N = xValues.shape[0]/32
HW = numpy.array([2.0*numpy.pi*i / (N-1) for i in range(0, N)])

# Do the convolution
hwConvolved = numpy.convolve(HW/HW.sum(), sinWithNoise, mode='same')

# Plot the data, and save as an image
plt.plot(xValues, sinWithNoise) # default colour is blue
plt.plot(xValues, hwConvolved, linewidth=2, color='red')
plt.xlabel("x")
plt.ylabel("y")
plt.savefig("filteredSinNoise.png", format='png')
plt.show()

```

Plots: The resulting plots



More resources

- An ebook about python: https://automatetheboringstuff.com/
- StackOverflow – the answer to pretty much any question is here: http://stackoverflow.com/questions/tagged/python
- StackOverflow Documentation – a Wikipedia-like documentation of python (and loads of other languages) http://stackoverflow.com/documentation/python/topics
- List of cool python libraries/scripts: https://github.com/vinta/awesome-python
- Numpy documentation: https://docs.scipy.org/doc/numpy/

See you in the practical!