

Python Basics

Andrew Green

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?

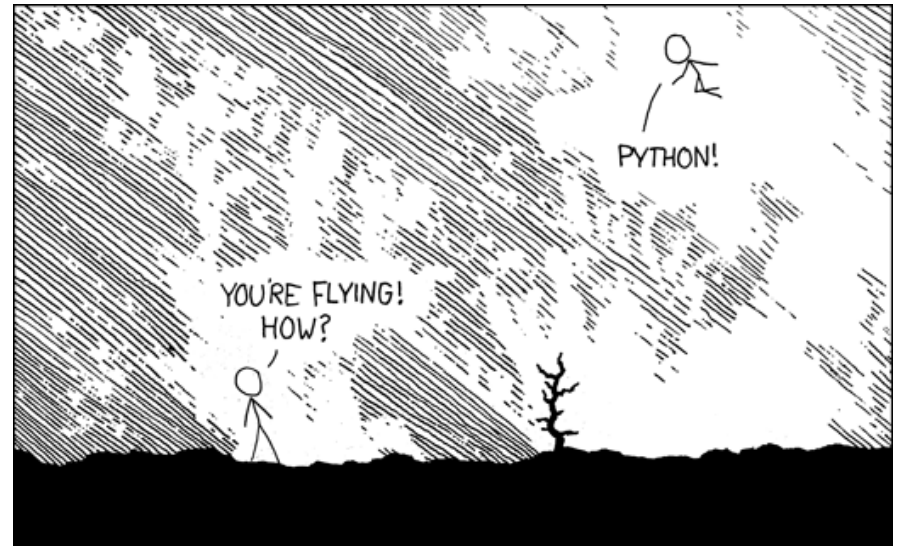
- There are lots of programming languages
 - C/C++
 - R
 - Brainf**k
- Python is relatively simple, but still very powerful.
- It's a 'gateway' language.

```
146 draw_dendrogram = function(hc, gaps, horizontal = T){
147   h = hc$height / max(hc$height) / 1.05
148   m = hc$merge
149   o = hc$order
150   n = length(o)
151
152   m[m > 0] = n + m[m > 0]
153   m[m < 0] = abs(m[m < 0])
154
155   dist = matrix(0, nrow = 2 * n - 1, ncol = 2, dimnames = list(NULL, c("x", "y")))
156   dist[1:n, 1] = 1 / n / 2 + (1 / n) * (match(1:n, o) - 1)
157
158   for(i in 1:nrow(m)){
159     dist[n + i, 1] = (dist[m[i, 1], 1] + dist[m[i, 2], 1]) / 2
160     dist[n + i, 2] = h[i]
161   }
162
163   draw_connection = function(x1, x2, y1, y2, y){
164     res = list(
165       x = c(x1, x1, x2, x2),
166       y = c(y1, y, y, y2)
167     )
168
169     return(res)
170   }
171
172   x = rep(NA, nrow(m) * 4)
173   y = rep(NA, nrow(m) * 4)
174   id = rep(1:nrow(m), rep(4, nrow(m)))
175
176   for(i in 1:nrow(m)){
177     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])
178     k = (i - 1) * 4 + 1
179     x[k : (k + 3)] = c$x
180     y[k : (k + 3)] = c$y
181   }
```

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

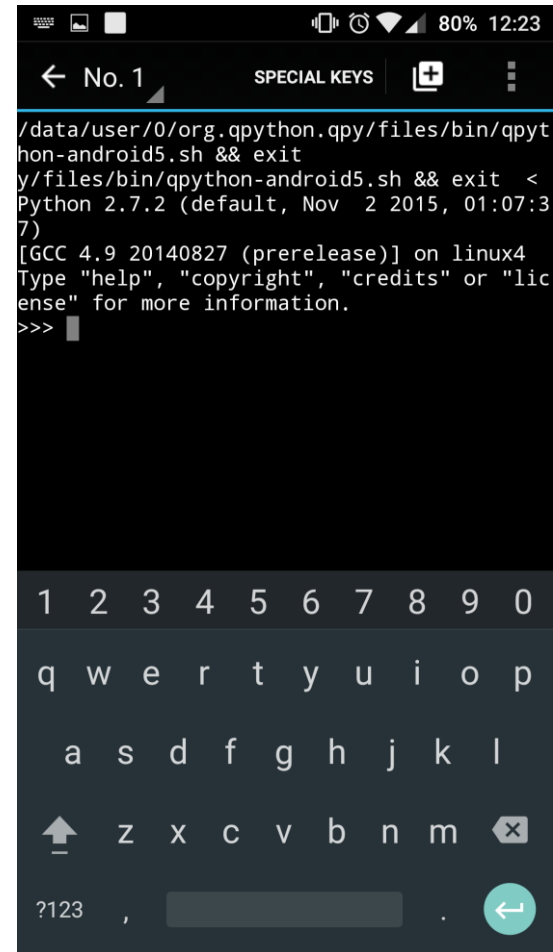
Python: Getting started

- 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



Python: Getting started

- You can even get python on your phone!
 - Might be useful for trying things?
- We will be using Anaconda python and Jupyter notebooks
 - You'll see it later



The screenshot shows a mobile terminal application interface. At the top, there's a status bar with icons for signal, battery (80%), and time (12:23). Below that, the app header includes a back arrow, the text 'No. 1', a 'SPECIAL KEYS' button, and a plus icon. The main area displays a terminal session where a user has executed a script to start a Python shell. The output shows the Python version (2.7.2) and the GCC version (4.9). The prompt is now '>>>'.

```
/data/user/0/org.qpython.qpy/files/bin/qpython-android5.sh && exit
y/files/bin/qpython-android5.sh && exit <
Python 2.7.2 (default, Nov 2 2015, 01:07:37)
[GCC 4.9 20140827 (prerelease)] on linux4
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

Python: Indentation

- 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...

```
138 def GenerateSettings():
139     """
140     Here we read the current settings JSON, and ask the user to verify all of it, then test the client_secrets file by attempting
141     if this fails, we instruct the user to get their client secrets file and re-run.
142     """
143     firstTime = False
144     try:
145         settings = json.loads(open(SETTINGS_FILE, 'r').read())
146     except:
147         print("Error: settings.json not found. Generating new from scratch...")
148         settings = {}
149         settings["compute"] = {}
150         settings["oauth_storage"] = "oauth2.dat" # A default value
151         firstTime = True
152
153     # Project ID
154     if "project" in settings:
155         print("The project ID in the current settings file is:\n\n%s\n\nIs this correct?"%(settings["project"]))
156         choice = raw_input("y/n? ")
157         if choice.upper() == "Y":
158             pass
159         else:
160             newproj = raw_input("Input the correct project ID:\t")
161             settings["project"] = newproj
162     else:
163         newproj = raw_input("Input the your project ID:\t")
164         settings["project"] = newproj
165
166     # Storage Bucket
167     if "bucket" in settings:
168         print("The bucket name in the current settings file is:\n\n%s\n\nIs this correct?"%(settings["bucket"]))
169         choice = raw_input("y/n? ")
170         if choice.upper() == "Y":
171             pass
172         else:
173             newBucket = raw_input("Input the correct bucket name:\t")
```


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 '#'

```
521 storage = Storage(settings['oauth_storage'])
522 credentials = storage.get()
523
524 # Authorize an instance of httplib2.Http. This only happens if we have not authorized, or if the token is invalid
525 if credentials is None or credentials.invalid:
526     flow = flow_from_clientsecrets(settings['client_secrets'], scope=settings['compute_scope'], redirect_uri='urn:ietf:oauth:2.0:oob')
527     auth_uri = flow.step1_get_authorize_url()
528     webbrowser.open_new(auth_uri)
529     auth_code = raw_input('Enter the auth code: ')
530     credentials = flow.step2_exchange(auth_code)
531
532 # Save the credentials so we don't need to authenticate again
533 storage.put(credentials)
534 http = httplib2.Http()
535 auth_http = credentials.authorize(http)
536
537 # Build services
538 computeService = build("compute", "v1", http=auth_http)
539 poolService = build("replicapool", "v1beta2", http=auth_http)
540 storageService = build("storage", "v1", http=auth_http)
541
542 # Register an emergency cleanup function incase things go south
543 atexit.register(emergency_cleanup, poolService=poolService, computeService=computeService, project=settings['project'],
544
545
546 # Use hardcoded URIs here rather than api queries - quite a bit faster
547 image_url = "{0}/{1}/global/images/{2}".format(URI_BASE, settings['compute']['image_project'], settings['compute']['image'])
548 zone_url = "{0}/{1}/zones/{2}".format(URI_BASE, settings['project'], settings['compute']['zone'])
549 mach_type_url = "{0}/{1}/zones/{2}/machineTypes/{3}".format(URI_BASE, settings['project'], settings['compute']['zone'], settings['compute']['mach_type'])
550 disk_type_url = "{0}/{1}/zones/{2}/diskTypes/{3}".format(URI_BASE, settings['project'], settings['compute']['zone'], settings['compute']['disk_type'])
551 networks_url = "{0}/{1}/global/networks/{2}".format(URI_BASE, settings['project'], settings['compute']['network'])
552
553
554
```

Python: Libraries

- 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.

```
1 """
2 My attempt to implement the Bragg Peak equation found in Bortfeld '97, Med Phys 24 (12) 2024-
3
4 Version 0.0.1a: Working implementation, by default we use the 'inaccurate' version of the dose c
5
6 """
7 import numpy as np
8 import matplotlib.pyplot as plt
9 import scipy.special as sp
10
11 toGray = 1.602E-10
12
13 def D(R0, phi0, epsilon, sig, z):
14     """
15     This is the very specialised equation for water only. It's equation 28/29 in the paper
16     """
17     if z < (R0 - 10*sig):
18         fac = (phi0)/(1.0 + 0.012*R0)
19         term1 = 17.93*((R0 - z)**-0.435)
20         term2 = ((0.444 + 31.7*epsilon)/R0)*((R0-z)**0.565)
21         return fac*(term1 + term2) * toGray
```

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!”
 - More on strings later...
 - 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`.

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

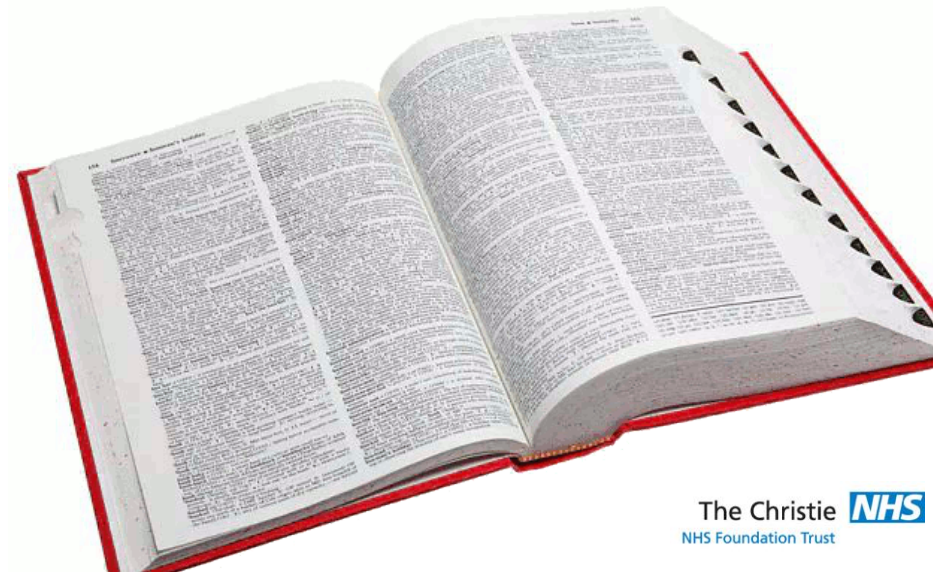
```
>>> a = list(1, 2, 3)
>>> b = [1, 2, 3]
>>> c = b.append(4)
>>> c
[1, 2, 3, 4]
>>> d = c.extend([5, 6, 7])
>>> d
[1, 2, 3, 4, 5, 6, 7]
```

- 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'
```



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 like the list type.
 - Some that I use a lot are
 - startswith()
 - endswith()
 - split()
 - strip()
 - Also useful is the join() method

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
```


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 = False
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)
```

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
0.333333333333
>>> 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
```

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 condition 1 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 in 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]`?

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

```
9 def readSerial(ser):  
10     # intValue = st.unpack(5*'c', ser.read(4))  
11     intValue = int(ser.readline().strip())  
12     return float(intValue)/255.0  
13  
14 N = 512  
15 t = np.arange(N)  
16 data = np.zeros((N,)) + 0.1
```

```
1 def sayHelloTo(thing):  
2     print("Hello" + thing + "!")  
3
```

```
31     data[i] = readSerial(ser)  
32     i += 1  
33     ii = [a % N for a in range(i+1, i+50)]  
34     data[ii] = 0.0  
35     i %= N  
36  
37  
38     aPlot.set_ydata(data)  
39     fig.canvas.draw()  
40     plt.pause(0.01)
```

Installing libraries

- There are 6000 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>
2
3  1  print("Hello, World!")
4
5
6
7
8
9
```

Example 2: Find all prime numbers below 100

```
1  currentNumber = 2
2  listOfPrimes = [2] # Need to start with 2 because *everything* is divisible by 1
3  while currentNumber < 100: # Only looking for numbers less than 100
4      isPrime = False
5      for p in listOfPrimes:
6          if currentNumber % p == 0: # See if the current number is divisible by any of our known primes
7              isPrime = False # If it is, it can't be prime!
8              break # No point checking the rest of the list
9          else:
10             isPrime = True # It might be prime - need to check the other numbers in our list
11             continue # Jump to the next iteration
12
13     if isPrime:
14         listOfPrimes.append(currentNumber) # Add the latest number to the primes list, if it is prime.
15         currentNumber += 1 # Increment the current number
16
17 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
```

ber

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 have CSV data
```

```
txtData = np.loadtxt("fileName.csv", delimiter=',')
```

```
# If you saved an array from a previous python run
```

```
npaData = 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!
```

Convolution

- If you don't know what convolution is, don't worry.
 - It will be covered tomorrow.
- Convolution in numpy is easy
 - Get your signal
 - Get your filter
 - Call `np.convolve(signal, filter (,options))`
 - Done
- Use the option `mode="same"` for filters

Example 4: Convolution of noisy data with a filter.

```
1 import numpy
2
3 # Create a 1024 length array of numbers between 0 and 2 pi
4 N = 1024
5 xValues = numpy.linspace(0, 2.0*numpy.pi, N, endpoint=True)
6
7 # make a noisy signal - sin^4 plus random gaussian noise
8 sinWithNoise = numpy.sin(xValues)**4 + numpy.random.normal(loc=0.0, scale=0.1, size=xValues.shape[0])
9
10 # Hamming window - used to do the filtering
11 alpha = 0.54
12 beta = 1.0 - alpha
13 N = xValues.shape[0]/32
14 HW = numpy.array([2.0*numpy.pi*i / (N-1) for i in range(0, N)])
15
16 # Do the convolution
17 hwConvolved = numpy.convolve(HW/HW.sum(), sinWithNoise, mode='same')
18
```

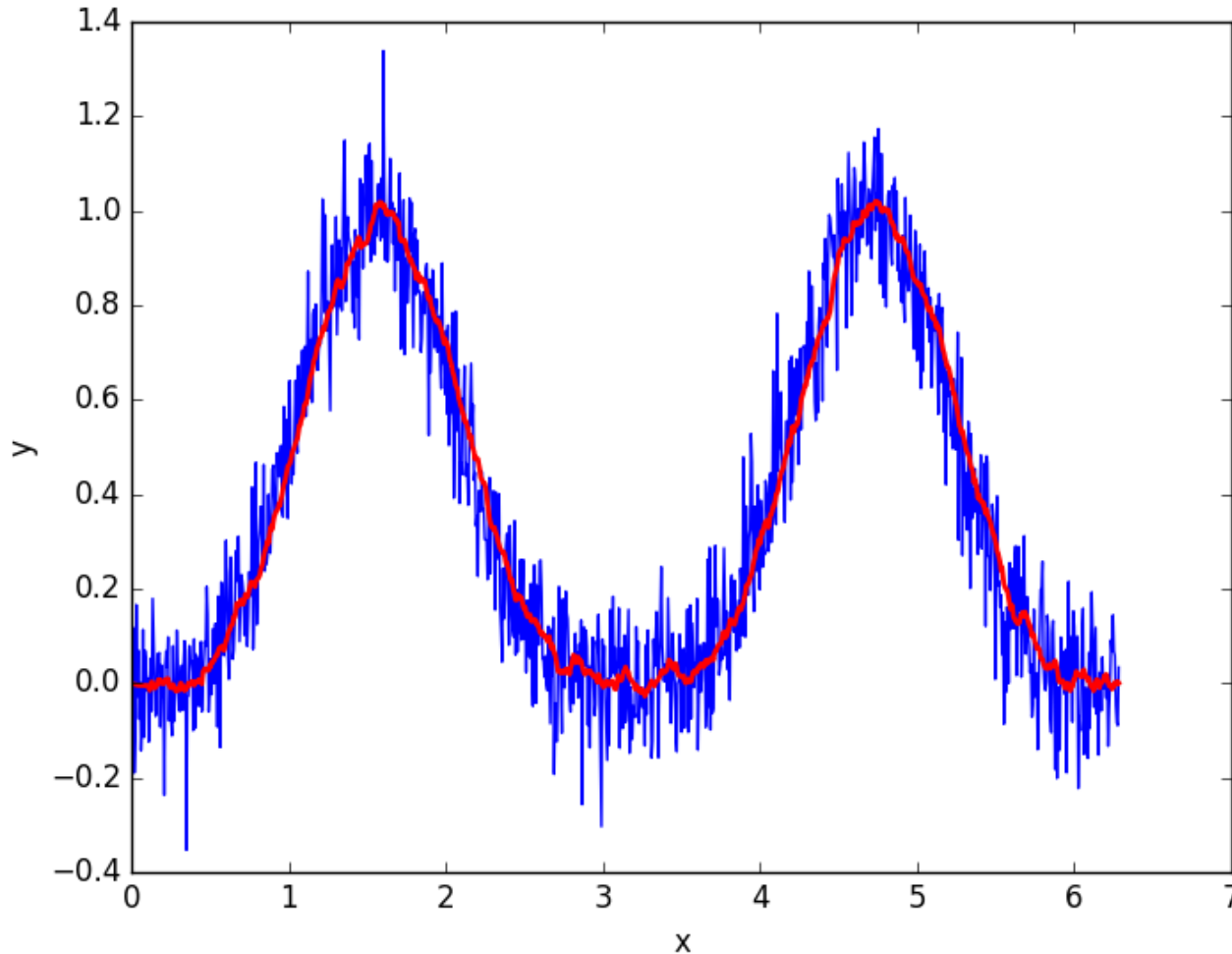
Python data analysis: plotting

- 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

```
1 import numpy
2 import matplotlib.pyplot as plt
3
4 # Create a 1024 length array of numbers between 0 and 2 pi
5 N = 1024
6 xValues = numpy.linspace(0, 2.0*numpy.pi, N, endpoint=True)
7
8 # make a noisy signal - sin^4 plus random gaussian noise
9 sinWithNoise = numpy.sin(xValues)**4 + numpy.random.normal(loc=0.0, scale=0.1, size=xValues.shape[0])
10
11 # Hamming window - used to do the filtering
12 alpha = 0.54
13 beta = 1.0 - alpha
14 N = xValues.shape[0]/32
15 HW = numpy.array([2.0*numpy.pi*i / (N-1) for i in range(0, N)])
16
17 # Do the convolution
18 hwConvolved = numpy.convolve(HW/HW.sum(), sinWithNoise, mode='same')
19
20 # Plot the data, and save as an image
21 plt.plot(xValues, sinWithNoise) # default colour is blue
22 plt.plot(xValues, hwConvolved, linewidth=2, color='red')
23 plt.xlabel("x")
24 plt.ylabel("y")
25 plt.savefig("filteredSinNoise.png", format='png')
26 plt.show()
27
```

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/>

Pro-tips for the practical

- Ask for help
 - It's why we're all there!
- If something doesn't work, **don't delete it!**
 - If needed, comment it to get past the problem
 - Ask someone to have a look at it
- Play around!
 - Only the last two days are assessed
 - Today is all about getting used to python

See you in the practical!