PythonAndMantid

October 25, 2015

```
In [1]: %matplotlib inline
```

1 Why Python?

2 Using the iPython

3 Data types

3.1 Basic data types

```
In [2]: integer_example=5
```

```
In [3]: float_example=5.0
```

```
In [4]: boolean_example=True
```

```
In [5]: string_example='python'
```

```
In [6]: string_example+5
```

```
        ---------------------------------------------------------------------------
    TypeError                                 Traceback (most recent call last)

        <ipython-input-6-8a238cac995e> in <module>()
    ----> 1 string_example+5


        TypeError: cannot concatenate 'str' and 'int' objects
```

- **to check the type, use `type` function**

```
In [7]: print type(float_example)
```

```
<type 'float'>
```

```
In [8]: #this is a comment
```

3.2 Converting between types

```
In [9]: x = 'The meaning of life is ... '
        answer = 42
        y = x + str(answer)
        print y
```

```
The meaning of life is ... 42
```

- **By default, print outputs a newline at the end. Use comma to print on the same line**

```
In [10]: x=5
         y=6
         print x, y
         print x
         print y
```

```
5 6
5
6
```

```
In [11]: print 1/2
```

```
0
```

```
In [12]: print 1./2
         print 1/2.
         print float(1)/float(2)
```

```
0.5
0.5
0.5
```

3.3 Sequence data types - lists

```
In [13]: list_example=[1,2,3,4,5]
```

- **create list of integers with `range(min,max,step)`**
- **note that the list starts with min, but it is less then max**

```
In [14]: print range(10,20,2)
```

```
[10, 12, 14, 16, 18]
```

- **Add elements with append**

```
In [15]: list_example.append("some string")
         print list_example
```

```
[1, 2, 3, 4, 5, 'some string']
```

- **Access elements by index, starting at 0**

```
In [16]: print list_example[0], list_example[3],list_example[-1]
```

```
1 4 some string
```

```
In [17]: a=range(10)
         print a
         a[5]=2
         print a
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 2, 6, 7, 8, 9]
```

- **Fancy indexing [start:stop:stride]**

```
In [18]: a=range(10)
         print a
         print a[0:3]
         print a[5:10]
         print a[5:-1]
         print a[5:]
         print a[:5]
         print a[::2]
         print a[1::2]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2]
[5, 6, 7, 8, 9]
[5, 6, 7, 8]
[5, 6, 7, 8, 9]
[0, 1, 2, 3, 4]
[0, 2, 4, 6, 8]
[1, 3, 5, 7, 9]
```

3.4 Sequence data types - tuples

- **immutable list**
- **useful when returning values from a function**

```
In [19]: lottery_numbers = (1,2,3,4,5,6)
         print lottery_numbers[0]     # prints 1
         print lottery_numbers[1:3]   # prints (2,3)
         # Assignment not allowed
         lottery_numbers[3] = 42      # gives error "TypeError: 'tuple' object does not support item as
```

```
1
(2, 3)


        ---------------------------------------------------------------------------
    TypeError                                 Traceback (most recent call last)

        <ipython-input-19-76b273095974> in <module>()
          3 print lottery_numbers[1:3]   # prints (2,3)
          4 # Assignment not allowed
    ----> 5 lottery_numbers[3] = 42      # gives error "TypeError: 'tuple' object does not support item


        TypeError: 'tuple' object does not support item assignment
```

3.5 Sequence data types - dictionaries

- **key, value pairs, not in a user defined order**

```
In [20]: empty_dict = {}      # Empty dictionary
         my_lookup = {'a' : 1, 'b' : 2} # A dictionary with two keys, each
                                        # mapped to the respective value
         print my_lookup
```

```
        print my_lookup.keys()
        print my_lookup.values()
```

```
{'a': 1, 'b': 2}
['a', 'b']
[1, 2]
```

```
In [21]: print my_lookup['b']  # prints  2
         print empty_dict['a']   # Results in "KeyError: 'a'"
```

```
2


        ---------------------------------------------------------------------------
    KeyError                                  Traceback (most recent call last)

        <ipython-input-21-60396b27b0c5> in <module>()
          1 print my_lookup['b']  # prints  2
    ----> 2 print empty_dict['a']   # Results in "KeyError: 'a'"


        KeyError: 'a'
```

```
In [22]: empty_dict['a'] = 1
         my_lookup['b'] = 3
         print empty_dict
         print my_lookup
```

```
{'a': 1}
{'a': 1, 'b': 3}
```

```
In [23]: del my_lookup['b']   # Removes the key/value pair with the specified key
         print my_lookup
         my_lookup.clear()   # Empties the dictionary
         print my_lookup
```

```
{'a': 1}
{}
```

3.6 Sequence data types - Common operations

```
In [24]: print empty_dict,a
         print len(empty_dict), len(a)
```

```
{'a': 1} [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
1 10
```

```
In [25]: my_lookup = {'a' : 1, 'b' : 2}
         print 1 in my_lookup
         print 'a' in my_lookup
         print 1 in a
```

```
False
True
True
```

```
In [26]: print 3 not in a
         print 15 not in a
```

```
False
True
```

4 Control structures

4.1 Comparison testing

```
In [27]: a=5
         print a==2 #equality
         print a!=2 #inequality
         print a<5
         print a>=4
```

```
False
True
False
True
```

- Combine boolean expressions with `and`/ `not`/ `or`

```
In [28]: print a<10 and a>0
```

```
True
```

4.2 if . . . elif . . . else

```
In [29]: if a<0:
             print "a < 0"
         elif a<10:
             print "0 <= a <10"
         else:
             print "a >= 10"
```

```
0 <= a <10
```

- after the boolean condition, the line has to end with a colon
- the block to be executed has to be indented, all lines with the same indentation
- `elif` and `else` are optional

4.3 for loop

```
In [30]: for run in [1,2,3,4]:
             print run
```

```
1
2
3
4
```

- the sequence after `in` can be any sequence type (list, dictionary, etc.)
- the for statement ends with colon
- the block to be executed has to be indented

4.3.1 `break` exits the loop

```
In [31]: nums = [1,2,3,-1,5,6]
         list_ok = True
         for i in nums:
             if i < 0:
                 list_ok = False
```

```
            break

    if list_ok == False:
        print 'The list contains a negative number'
```

```
The list contains a negative number
```

4.3.2 `continue` jumps execution to the next iteration of the for loop

```
In [32]: nums =  [1,2,3,-1,5,6]
         pos_sum = 0
         for i in nums:
             if i < 0:
                 continue
             pos_sum += i      # compound assignment means pos_sum = pos_sum + i

         print 'Sum of positive numbers is ' + str(pos_sum)
```

```
Sum of positive numbers is 17
```

- **An optional `else` clause can be added after the loop that will only get executed if the whole loop executes successfully**

```
In [33]: for i in range(0,10):
             print i
         else:
             print 'done'      # Prints numbers 0-9 and the 'done'

         for i in range(0,10):
             if i == 5:
                 break
             print i
         else:
             print 'done'      # Prints numbers 0-4
```

```
0
1
2
3
4
5
6
7
8
9
done
0
1
2
3
4
```

- **one can use `enumerate` to get the index of the loop**

```
In [34]: day=['Mo','Tu','We','Th','Fr','Sa','Su']
         for index,d in enumerate(day):
             print 'Day ',index,' is ',d
```

```
Day  0  is  Mo
Day  1  is  Tu
Day  2  is  We
Day  3  is  Th
Day  4  is  Fr
Day  5  is  Sa
Day  6  is  Su
```

- similarly, one can tie together sequences of the same length, using `zip`

```
In [35]: alist=range(2,7)
         blist=range(5,10)
         print len(alist), len(blist)
         for x,y in zip(alist,blist):
             print x,y
```

```
5 5
2 5
3 6
4 7
5 8
6 9
```

4.4 while

```
In [36]: sum = 0
         while sum < 10:
             sum += 1    # ALWAYS remember to update the loop test or it will
                         # run forever!!

         print sum       # Gives value 10
```

```
10
```

- same syntax as for
- executes until the statement is `False`

Exercise 1: Write a program to generate a list of the 25 first Fibonacci numbers. Create an empty list. Append elements 0 and 1. Then write a loop to append fib[i]=fib[i-1]+fib[i-2]

5 Functions

```
In [37]: def sayHello():
             print ' ----- HELLO !!! ----- '
```

- starts with `def` keyword, function name, list of arguments in parantheses, and a colon
- the code inside the function is indented
- can return some values

```
In [38]: sayHello()
```

```
----- HELLO !!! -----
```

```
In [39]: print sayHello()
```

```
----- HELLO !!! -----
None
```

```
In [40]: type(sayHello())

----- HELLO !!! -----

Out[40]: NoneType
```

5.1 Function arguments

```
In [41]: def printSquare(n, verbose):
             if verbose == True:
                 print 'The square of ' + str(n) + ' is: ' + str(n*n)
             elif verbose == False:
                 print str(n*n)
             else:
                 print 'Invalid verbose argument passed'

         printSquare(2, True)  # Produces long string
         printSquare(3, False) # Produces short string
         printSquare(3,5)      # Produces error message

The square of 2 is: 4
9
Invalid verbose argument passed
```

- **Instead of positional arguments, we can use keyword arguments, where order is not important**
- **One can use a mixture of positional and keyword arguments, but all positional arguments must be placed before keyword arguments**

```
In [42]: printSquare(verbose = True, n = 2)  # produces the same as printSquare(2, True)

The square of 2 is: 4

In [43]: def foo(A,B,C,D,E):
             print A,B,C,D,E

In [44]: foo(1,2,3,4,5)      # Correct, no names given

1 2 3 4 5

In [45]: foo(1,2,3,D=4,E=5)  # Correct as the first 3 get assigned to the first
                             # 3 of the function and then the last two are
                             # specified by name

1 2 3 4 5

In [46]: foo(C=3,1, 2,4,5)   # Incorrect and will fail as a name has been
                             # specified first but then Python doesn't know
                             # where to assign the rest


          File "<ipython-input-46-2580d4d7f5ef>", line 1
        foo(C=3,1, 2,4,5)   # Incorrect and will fail as a name has been
    SyntaxError: non-keyword arg after keyword arg
```

- **one can use default values for arguments**

```
In [47]: def printSquare(n, verbose = False):
             if verbose == True:
                 print 'The square of ' + str(n) + ' is: ' + str(n*n)
             elif verbose == False:
                 print str(n*n)
             else:
                 print 'Invalid verbose argument passed'

         printSquare(2)                  # Produces short message
         printSquare(2, verbose = True)  # Produces long message
```

```
4
The square of 2 is: 4
```

- **Advanced topic: using lists and dictionaries as arguments**

```
In [48]: l=range(5)
         foo(*l)
```

```
0 1 2 3 4
```

```
In [49]: d={'A':3,'B':5,'C':9,'D':10,'E':0}
         foo(**d)
```

```
3 5 9 10 0
```

```
In [50]: newd={'D':1,'E':2}
         foo(1,2,3,**newd)
```

```
1 2 3 1 2
```

```
In [51]: foo(1,2,3,**d)
```

```
        ---------------------------------------------------------------------------
    TypeError                                 Traceback (most recent call last)

        <ipython-input-51-1ed12194d077> in <module>()
    ----> 1 foo(1,2,3,**d)


        TypeError: foo() got multiple values for keyword argument 'A'
```

5.2 Return values

```
In [52]: def square(n):
             return n*n

         two_squared = square(2)
         # or print it as before
         print square(2)
```

```
4
```

- **functions can return multiple values**

```
In [53]: def square(x,y):
             return x*x, y*y

         t = square(2,3)
         print t  # Produces (4,9)
         # Now access the tuple with usual operations
```

```
(4, 9)
```

- **the output values can be unpacked directly, similarly to the `enummerate` example**

```
In [54]: def square(x,y):
             return x*x, y*y

         xsq, ysq = square(2,3)
         print xsq  # Prints 4
         print ysq  # Prints 9
```

```
4
9
```

Exercise 2: Starting from the previous program, write a function to return the Fibonacci numbers. The function should contain two parameters: the numbers to be returned (default 20), and a boolean flag (default false). If the flag is false, just return the list of Fibonacci numbers. If the flag is true, return two lists. The first is the Fibonacci numbers, the second is the ratio of Fibonacci numbers fib[i]/fib[i-1]. The list should be empty if n<=2, and it should not contain 1/0 as the first element.

6 Modules

- **collection of functions and abjects in a library (a file with .py extension)**
- **functions must be imported**
- **avoid name conflicts (same function name defined in different modules)**

```
In [55]: ## File: mymath.py ##
         def square(n):
             return n*n

         def cube(n):
             return n*n*n

         def average(values):
             nvals = len(values)
             sum = 0.0
             for v in values:
                 sum += v
             return float(sum)/nvals
```

```
In [56]: ## My script using the math module ##
         import mymath  # Note no .py

         values = [2,4,6,8,10]
         print 'Squares:'
         for v in values:
             print mymath.square(v)
         print 'Cubes:'
         for v in values:
             print mymath.cube(v)

         print 'Average: ' + str(mymath.average(values))
```

```
        ---------------------------------------------------------------------------
    ImportError                               Traceback (most recent call last)

        <ipython-input-56-8297549e15b7> in <module>()
          1 ## My script using the math module ##
    ----> 2 import mymath  # Note no .py
          3
          4 values = [2,4,6,8,10]
          5 print 'Squares:'


        ImportError: No module named mymath
```

```
In [57]: import mymath as mt

         print mt.square(2)
         print mt.square(3)
```

```
        ---------------------------------------------------------------------------
    ImportError                               Traceback (most recent call last)

        <ipython-input-57-0717d9a057bc> in <module>()
    ----> 1 import mymath as mt
          2
          3 print mt.square(2)
          4 print mt.square(3)


        ImportError: No module named mymath
```

```
In [58]: from mymath import *
         average(values)
```

```
        ---------------------------------------------------------------------------
    ImportError                               Traceback (most recent call last)

        <ipython-input-58-72dfc17ceda1> in <module>()
    ----> 1 from mymath import *
          2 average(values)


        ImportError: No module named mymath
```

- **python is looking for modules in the current directory, and in a specified list of folders, specified by the PYTHONPATH variable**

```
In [59]: import sys
         print sys.path
```

```
['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-x86_64-linux-gnu', '/usr/lib/python2.7/lib-tk', '/us
```

- **`sys.path` is a list, and you can append or insert the path to your module in it**

6.1 Python Standard Library https://docs.python.org/2/library/

- **datetime**

```
In [60]: import datetime as dt
         format = '%Y-%m-%dT%H:%M:%S'
         t1 = dt.datetime.strptime('2008-10-12T14:45:52', format)
         print 'Day', t1.day
         print 'Month',t1.month
         print 'Minute',t1.minute
         print 'Second',t1.second

         # Define todays date and time
         t2 = dt.datetime.now()
         diff = t2 - t1
         print 'How many days difference?',diff.days
```

```
Day 12
Month 10
Minute 45
Second 52
How many days difference? 2568
```

- **os.path**

```
In [61]: import os.path

         directory = 'C:/Users/Files'
         file1 = 'run1.txt'
         fullpath = os.path.join(directory, file1)  # Join the paths together in
                                                    # the correct manner

         # print stuff about the path
         print os.path.basename(fullpath)  # prints 'run1.txt'
         print os.path.dirname(fullpath)  # prints 'C:\Users\Files'

         # A userful function is expanduser which can expand the '~' token to a
         # user's directory (Documents and Settings\username on WinXP  and
         # /home/username on Linux/OSX)
         print os.path.expanduser('~/test') # prints /home/[MYUSERNAME]/test on
                                            # this machine where [MYUSERNAME] is
                                            # replaced with the login
```

```
run1.txt
C:/Users/Files
/home/3y9/test
```

```
In [62]: print os.path.isdir('/SNS/users/shared')
         print os.path.isfile('/path/to/nowhere.txt')
```

```
True
False
```

- **math (note- we will use numpy later instead)**

```
In [63]: import math
         print math.sin(3.)
         print math.log(42.)
         print math.degrees(math.pi)
```

```
0.14112000806
3.73766961828
180.0
```

- **glob - returns pathnames matching pattern**

```
In [64]: import glob
         print glob.glob('/SNS/users/shared/*')
```

```
['/SNS/users/shared/MantidTrainingCourseData', '/SNS/users/shared/SasViewTrainingCourseData']
```

Exercise 3: In the mantid training course directory there are several text files. Create a dictionary containing as keys the name of the files, without paths or extensions, and as values the size of the files. Use glob, to search for the files with txt extension, then os.path.basename, os.path.splitext, and os.path.getsize

7 Numpy module http://docs.scipy.org/

- **numerical module for arrays - similar lo lists, but all elements must be the same type**
- **designed for fast computation**

```
In [65]: import numpy as np
         a=np.array([1,2,3])
         print a
         print a**2
         print type(a)
         print a.shape
         print len(a)
         b=np.array([[1,2,3],[4,5,6]])
         print b
         print b.shape
```

```
[1 2 3]
[1 4 9]
<type 'numpy.ndarray'>
(3,)
3
[[1 2 3]
 [4 5 6]]
(2, 3)
```

- **other array creation routines**

```
In [66]: print np.arange(1,7,2) #start, stop(exclusive), step
         print np.arange(4)
```

```
[1 3 5]
[0 1 2 3]
```

```
In [67]: print np.linspace(0,np.pi,10) #min,max, number of steps (default 50)
```

```
[ 0.          0.34906585  0.6981317   1.04719755  1.3962634   1.74532925
  2.0943951   2.44346095  2.7925268   3.14159265]
```

```
In [68]: print np.ones((3,3))
```

```
[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
```

```
In [69]: print np.eye(3)
```

```
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

```
In [70]: print np.zeros((2,3))
```

```
[[ 0.  0.  0.]
 [ 0.  0.  0.]]
```

- **indexing is similar to the lists**

```
In [71]: print b[1,:]
```

```
[4 5 6]
```

- **statistical and mathematical functions**

```
In [72]: a=np.arange(6)
         print a
         print a+2
         print a.mean()
         print a.sum()
         print np.sin(np.pi*a/4)
```

```
[0 1 2 3 4 5]
[2 3 4 5 6 7]
2.5
15
[  0.00000000e+00   7.07106781e-01   1.00000000e+00   7.07106781e-01
   1.22464680e-16  -7.07106781e-01]
```

```
In [73]: x=np.linspace(-np.pi,np.pi,10000)
         y=np.sin(x)
         print 'Maximum value of y is ',y.max(), ' found at x = ',x[y.argmax()]/np.pi,'*pi'
```

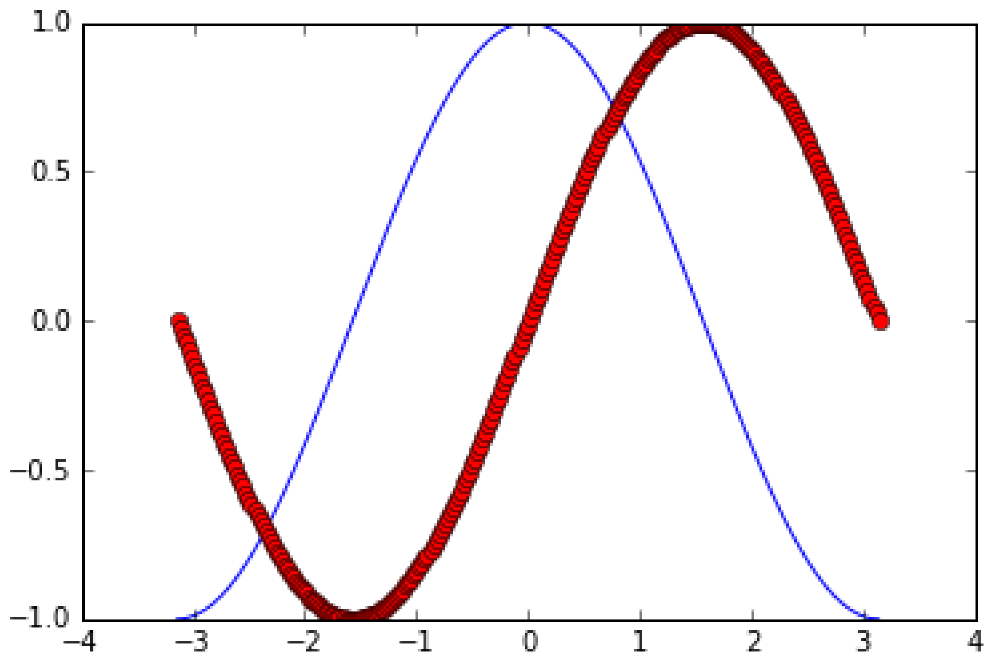
```
Maximum value of y is  0.999999987661  found at x =  0.499949994999 *pi
```

- **note that math also contain a sin function, that does not act on arrays. Numpy functions can deal with floats**

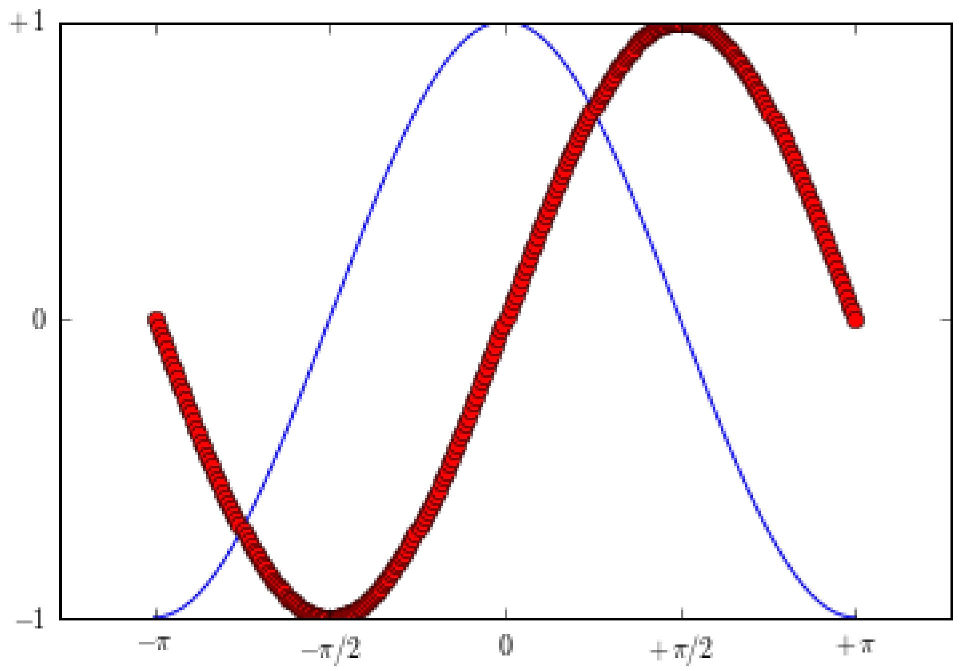
8 Matplotlib module - http://matplotlib.org/

- module containg plotting routines
- `matplotlib.pyplot` modeled to resemble Matlab

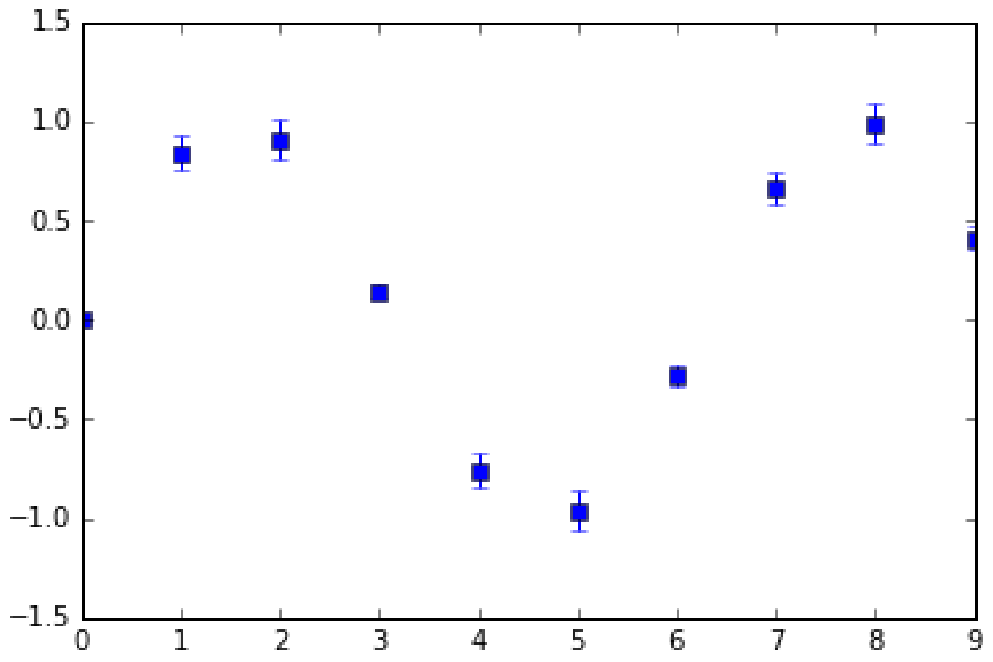
```
In [74]: import numpy as np
         import matplotlib.pyplot as plt

         X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
         C, S = np.cos(X), np.sin(X)

         plt.plot(X, C)
         plt.plot(X, S,'ro')

         plt.show()
```



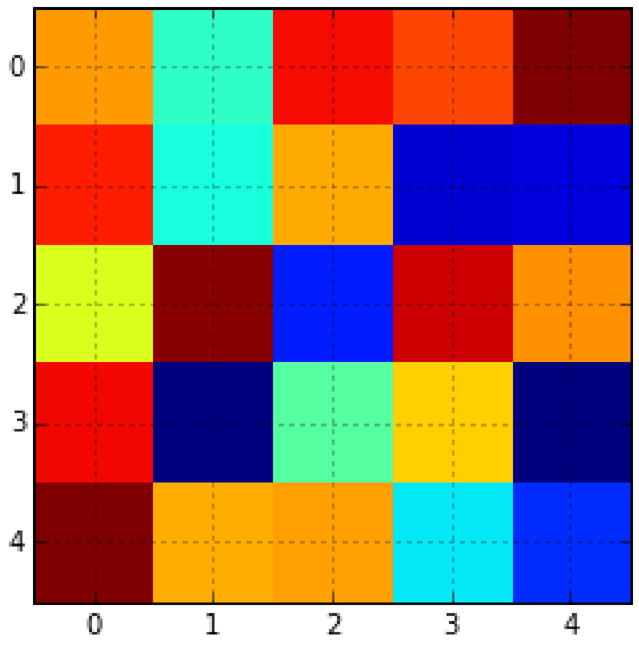
```
In [75]: import numpy as np
         import matplotlib.pyplot as plt

         X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
         C, S = np.cos(X), np.sin(X)

         plt.plot(X, C)
         plt.plot(X, S,'ro')

         plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
                   [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])

         plt.yticks([-1, 0, +1],
              [r'$-1$', r'$0$', r'$+1$'])
plt.show()
```



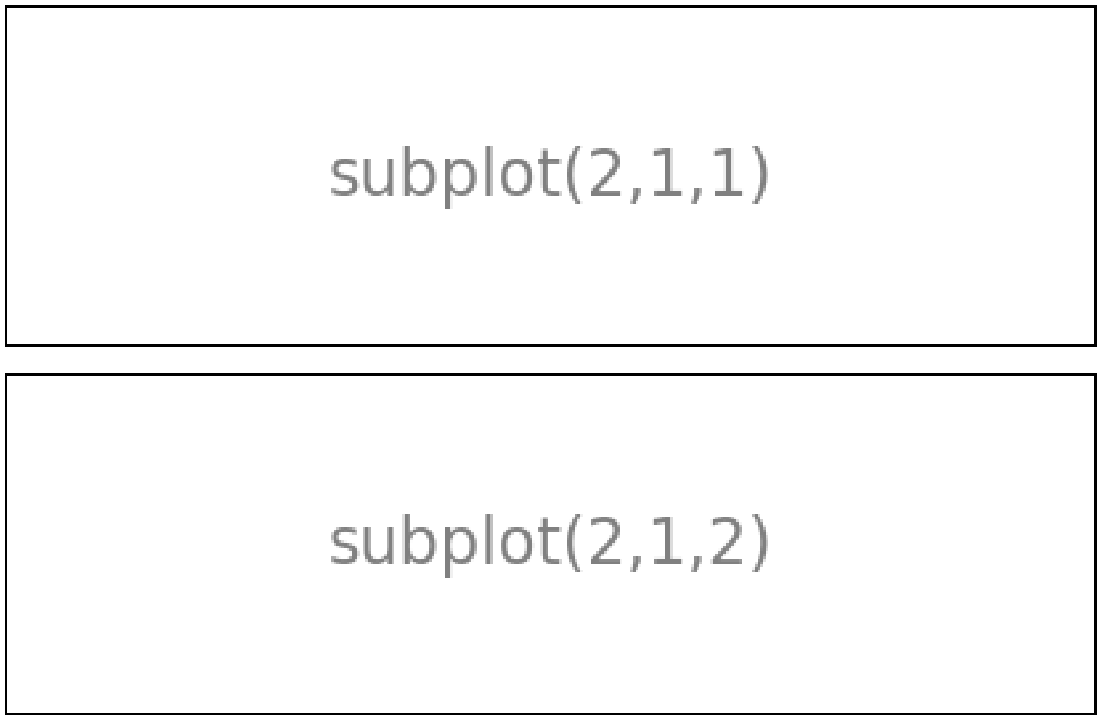
```
In [76]: import numpy as np
         import matplotlib.pyplot as plt

         X=np.arange(10)
         Y=np.sin(X)
         err=np.sqrt(np.abs(Y))*0.1
         plt.errorbar(X,Y,err,fmt='bs')
         plt.show()
```



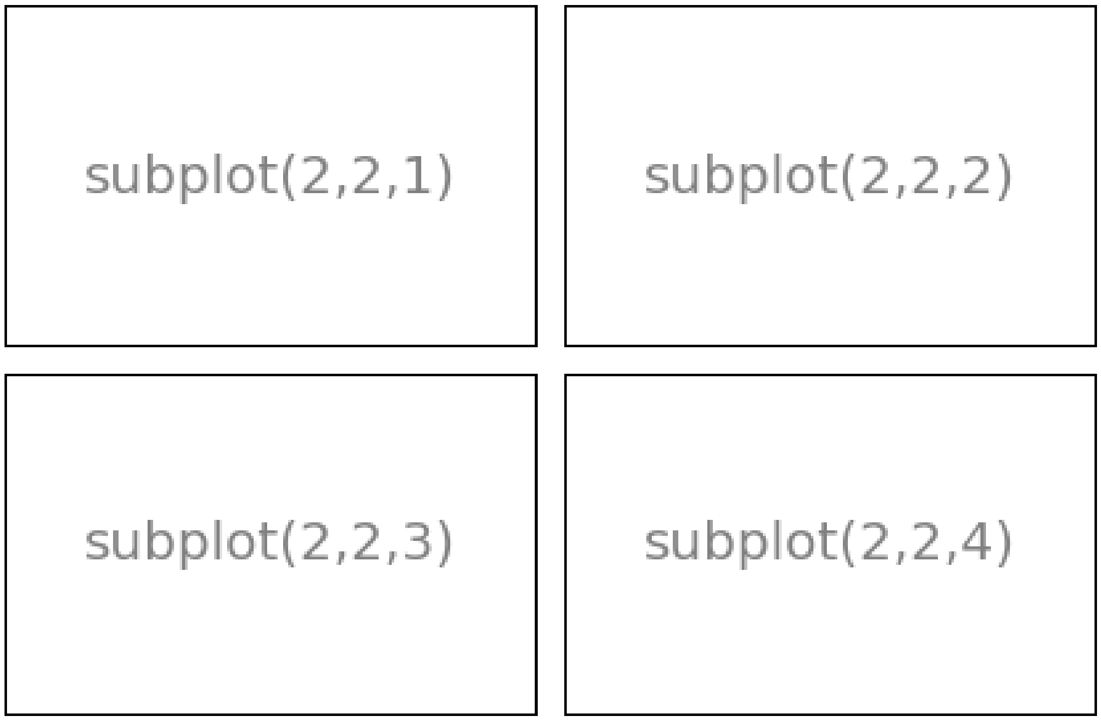
```
In [77]: import numpy as np
         import matplotlib.pyplot as plt

         A=np.random.rand(5,5)
         plt.imshow(A, interpolation='nearest')
         plt.grid(True)
         plt.show()
```



```
In [78]: import matplotlib.pyplot as plt

         plt.figure(figsize=(6, 4))
         plt.subplot(2, 1, 1)
         plt.xticks(())
         plt.yticks(())
         plt.text(0.5, 0.5, 'subplot(2,1,1)', ha='center', va='center',
                 size=24, alpha=.5)

         plt.subplot(2, 1, 2)
         plt.xticks(())
         plt.yticks(())
         plt.text(0.5, 0.5, 'subplot(2,1,2)', ha='center', va='center',
                 size=24, alpha=.5)

         plt.tight_layout()
         plt.show()
```



```
In [79]: import matplotlib.pyplot as plt

         plt.figure(figsize=(6, 4))
         plt.subplot(2, 2, 1)
         plt.xticks(())
         plt.yticks(())
         plt.text(0.5, 0.5, 'subplot(2,2,1)', ha='center', va='center',
                 size=20, alpha=.5)

         plt.subplot(2, 2, 2)
         plt.xticks(())
         plt.yticks(())
         plt.text(0.5, 0.5, 'subplot(2,2,2)', ha='center', va='center',
                 size=20, alpha=.5)

         plt.subplot(2, 2, 3)
         plt.xticks(())
         plt.yticks(())

         plt.text(0.5, 0.5, 'subplot(2,2,3)', ha='center', va='center',
                 size=20, alpha=.5)

         plt.subplot(2, 2, 4)
         plt.xticks(())
         plt.yticks(())
         plt.text(0.5, 0.5, 'subplot(2,2,4)', ha='center', va='center',
                 size=20, alpha=.5)

         plt.tight_layout()
         plt.show()
```



Exercise 4: Plot the function $y = x^3 - 2x^2 - 19x + 20$, from -5 to 7, using 1000 points. Show the grid. You can guess that the roots of this polynomial are -4, 1 and 5. To check, use these values as initial guesses, generate 200 points in a range ± 2 around the guesses, calculate y, find the index where $|y|$ is minim, and the corresponding x is the solution. Try with a different number of generated points, both odd and even.

Exercise 5: Plot the ratio of Fibonacci numbers generated using the function from a previous exercise. The x axis is the index of the corresponding large number. Choose the symbol to be red squares. Show that the converge towards the golden ratio $(1+\sqrt{5})/2$***, by plotting a blue line from x=0 to x=20. Add some labels to the axes.***

9 Working with ASCII files

- **To be able to read/write a file we must first open a "handle" to it through the open command**
- **When done with reading/writing, close the file**

```
In []: handle = open('filename.txt', 'r')
```

- **The flags are**

1. read - 'r'
2. write - 'w'
3. append - 'a'
4. and binary - 'b'.

```
In []: contents = handle.read() #reads everything into one big chunk
       for line in hadle: #reads line by line
           print line
```

- **Note that in the example above the line contains a newline character. You can either strip it with `line.rstrip()` or add a comma after `print line`**

```
In []: handle.close()
```

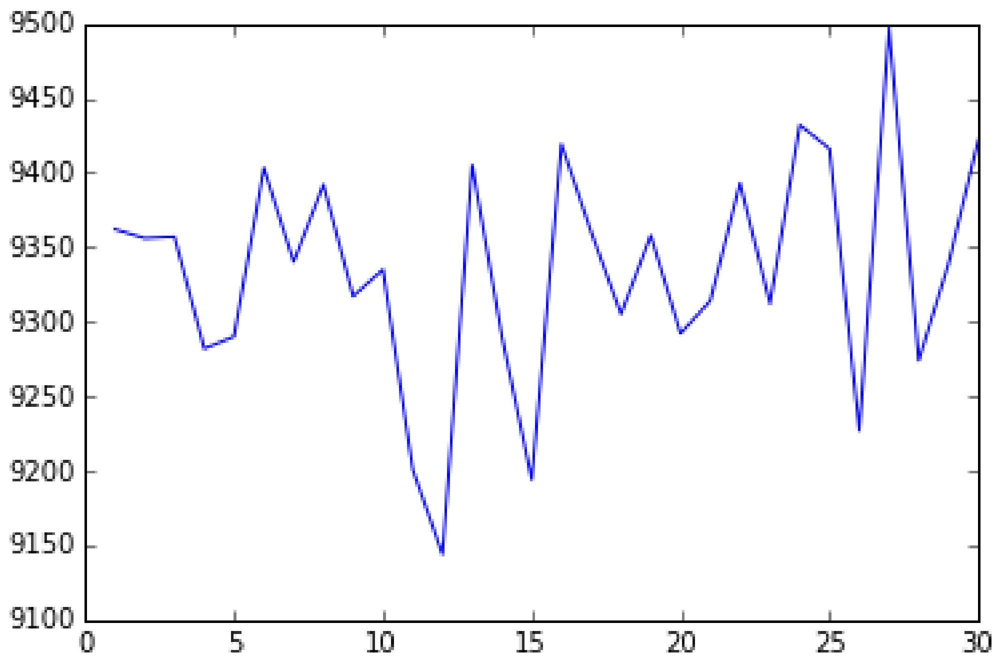
- **To write a string to a file, use the `write()` function**

```
In []: handle = open('NewFile.txt', 'w')
       handle.write('1 2 3 4 5 6\n') #note the newline. write does not put it automatically
       handle.write('7 8 9 10 11\n')
       handle.close()

       # Produces a file with the numbers on 2 separate lines
```

Exercise 6: Choose one of the text files from exercise 3. Read each line as a string, extract the value on the second column, and then calculate the average of all these values.

- **For nicely formatted ASCII files, there is a numpy function to read them**

```
In [81]: import numpy as np
         data=np.loadtxt('/SNS/users/shared/MantidTrainingCourseData/HB2A_exp0440_scan0001.dat')
         import matplotlib.pyplot as plt
         plt.plot(data[:-1,0],data[:-1,3])

Out[81]: [<matplotlib.lines.Line2D at 0x7f966d169c90>]
```



10 Generating Mantid scripts

- In MantidPlot, you can run several algorithms, then look at the history of a workspace. You can then generate a python script that can be reused
- If you want to be able to run the cript from outside MantidPlot, you need to tell python where to find the Mantid libraries, and then import those libraries. For simply running algolrithms, import everything from `mantid.simpleapi`

Exercise 7: Load SEQUOIA run 17880, sum spectra, rebin with a step of 20. Generate the script, add all the necessary stuff to run it from outside MantidPlot. Expand the script (use a for loop) to do the processing of runs 17880 through 17883. Make sure the workspace names are different. Save each workspace in a file in your home directory. Open these files in MantidPlot, and check if they look reasonable (each of them should have one spectra, and all of them should have a peak at about the same time of flight value

11 Accessing workspaces

- In the script generated before, the algorithms are using workspaces names. In order to access worspace properties, and to use workspace arithmetic, one needs to get the workspace *handle*.

Method 1: as a return value of an algorithm call

```
In []: w=Load('SEQ_17880') #Note that there is no need to use the OutputWorkspace property. The output
```

Note that some algorithms return a tuple, and your workspace is only one of the elements. See for example GetEi

Method 2: using mtd['workspace_name']

```
In []: Load('SEQ_17880',OutputWorkspace='awesomeData')
       w=mtd['awesomeData'] #note that the handle name is different than the workspace name
```

Note that if the name does not exists, python will produce an error To find out all workspace names that are available, use the `mtd.getObjectNames()`* **command***

Method 3: using mtd.importAll(). This will generate a handle for each workspace name. The handle name is the same as the workspace name, unless an invalis handle name would result. In that case, a "cleaned" name is generated. If the workspace name starts with a number, an underscore will be added at the beginning. All non alphanumeric characters will be replaced by undersocres

- **Workspace algebra**

```
In []: w1 = mtd['workspace1']
       w2 = mtd['workspace2']

       # Sum the two workspaces and place the output into a third
       w3 = w1 + w2

       # Multiply the new workspace by 2 and place the output into a new workspace
       w4 = w3 * 2

       # Multiply a workspace by 2 and replace w1 with the output
       w1 *= 2.0

       # Add 'workspace2' to 'workspace1' and replace 'workspace1' with the output
       w1 += w2
```

- **Reading/writing data**

For matrix workspaces, one can use readX(index),readY(index),readE(index) to read X,E, and E values. These functions return a numpy array of values.
To find out how many histograms are in a workspace, use `getNumberHistograms()` **function.**
One can find the number of bins with the `blocsize()` **function.**

```
In []: # loop access. Print the last value in all spectra
       for index in range(0, raw_workspace.getNumberHistograms()):
           #Note the round brackets followed by the square brackets
           print raw_workspace.readY(index)[raw_workspace.blocksize()-1] #we could have just used [-1]
```

extractX(), extractY(), and extractE() methods return numpy arrays with the shape given by (getNumberHistograms, blocksize) or (getNumberHistograms, blocksize+1) (for X)

```
In []: print w.extractY().shape
```

dataX, dataY, and dataE return read/write numpy arrays. Note that you cannot write data into an EventWorkspace

```
In []: xdata=w.dataX(0)
       xdata+=16666.67
       ydata=w.dataY(0)
       ydata=xdata[:-1]
```

For MDHistoWorkspaces the read write methods are getSignalArray, getErrorSquaredArray, getNumEventsArray, setSignalArray, setSignalAt, . . .

- **Getting information about sample logs**

All the log values are stored in the Run object. This is a dictionary with log names as keys. The values are dependent on the type of log value to be read

```
In [82]: import sys
         sys.path.append('/home/3y9/Mantid/Build/bin')
         from mantid.simpleapi import *
         w=Load('/SNS/users/shared/MantidTrainingCourseData/CNCS_7860_event.nxs')
         run=w.getRun()
         print run.keys()
```

```
['ChopperStatus1', 'ChopperStatus2', 'ChopperStatus3', 'ChopperStatus4', 'ChopperStatus5', 'CurrentSP',
```

To get a particular log, use the `.value` function

```
In [83]: print run['Filename'].value
         print run['Phase1'].value
```

```
/SNS/users/shared/MantidTrainingCourseData/CNCS_7860_event.nxs
[ 8798.984375    8798.98828125  8798.99609375  8798.99316406  8798.98730469
  8798.99707031  8798.99316406  8798.98242188  8798.98632812  8799.00390625
  8799.00292969  8798.98242188  8798.98535156  8798.984375    8798.97851562
  8798.97265625  8798.97558594  8798.97363281  8798.9765625   8798.97460938
  8798.98242188  8798.984375    8798.9921875   8798.98242188  8798.99707031
  8798.98339844  8798.97949219  8798.98632812  8798.98828125  8798.99316406
  8798.99804688  8799.00097656  8798.98632812  8798.98535156  8798.99023438
  8798.99121094  8799.00292969  8799.01464844  8799.0078125   8799.00097656
  8798.98925781  8798.99707031  8798.9921875   8798.99121094  8798.98144531
  8798.97851562]
```

For time series logs, one can also get the times, or statistics about the log

```
In [84]: print run['Phase1'].times
```

```
[2010-Mar-25 16:09:27.780000000,2010-Mar-25 16:09:30.108000068,2010-Mar-25 16:09:32.014000205,2010-Mar-
```

```
In [85]: print run['Phase1'].getStatistics().minimum
         print run['Phase1'].getStatistics().mean
         print run['Phase1'].timeAverageValue()
```

```
8798.97265625
8798.98904552
8798.98910967
```

To get the time in seconds from the beginning of the run, use something like this:

```
In [86]: phase1=w.getRun()['Phase1']
         times=[]
         for t in phase1.times:
             times.append((t-phase1.times[0]).total_seconds())
         print times
```

```
[0, 2, 4, 6, 9, 15, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47, 54, 56, 59, 62, 65, 68, 71, 74, 77, 80,
```

- **Getting information about the instrument**

The instrument handle is given by the getInstrument() function. The Instrument object has several components. You can get the number of components using the nelements function. To get the type of a component, just use the type function. Any ComponentAssembly objects have the nelements function. Detectors have the type `mantid.geometry._geometry.Detector`

All instruments/components have a `getName()` function

```
In [87]: inst=w.getInstrument()
         print inst.getName()," has ",inst.nelements(), " components"
         for i in range(inst.nelements()):
             print inst[i].getName(), type(inst[i])
```

```
CNCS  has  4  components
moderator <class 'mantid.geometry._geometry.ObjComponent'>
sample-position <class 'mantid.geometry._geometry.ObjComponent'>
monitors <class 'mantid.geometry._geometry.CompAssembly'>
detectors <class 'mantid.geometry._geometry.CompAssembly'>
```

```
In [88]: print inst[3][0].getName()
         print inst[3][0][0].getName()
         print inst[3][0][0][0].getName()
         print inst[3][0][0][0][0].getName()
```

```
bank1
eightpack
tube1
pixel1
```

For any instrument component one can get the position. For detectors one might be interested in getting the theta and phi angles, or the detector number

```
In [89]: from mantid.kernel import V3D
         det=inst[3][0][0][0][0]
         print det.getPos()
         print det.getID()
         print det.getTwoTheta(V3D(0,0,0),V3D(0,0,1)) #this value is in radians
```

```
[2.57229,-0.992187,-2.3661]
0
2.28003974119
```

Exercise 8:

Part A: from the previous exercise, get the handles to the workspaces using mtd.importAll(). Get the number of bins in the first workspace. Get the first spectrum data, and find out the maximum value, and the position of this value. Remember that the x axis contains bin boundaries. Plot the spectra from all workspaces in a single figure, with log scaling for y axis. Append the arrays for the x and y coordinates for each workspace, and create a new workspace, with 4 spectra. Use the CreateWorkspace algorithm. Use the return value of the CreateWorkspace algorithm as the handle for the new workspace. Change the X coordinate of spectra index 2 by adding 1000 to it. Save this workspace to a nexus file, and open it in MantidPlot. Plot all spectra.

Part B: Find out where is the moderator for SEQUOIA instrument. Print all component names and types for the childrens of the instrument. Write a script to find out how many banks are in each row. Get the detector in bankB1, tube1, pixel1, and print detector ID and theta (in degrees)

Part C: for the last file, extract and plot Phase2 log value, as a function of time, and get statistics about minimum, maximum, average, and time average

12 Information about mantid modules:

12.1 http://docs.mantidproject.org/nightly/api/python/mantid/index.html

- **mantid.kernel contains V3D and VMD objects, validators and properties for writing algorithms, and the logger object**

- mantid.geometry contains all instrument component objects, the goniometer, sample unit cell and orientation
- mantid.api includes all workspace objects, the run object, the sample object, and algorithms as objects(not individual algorithms)
- mantid.simpleapi is made up of all algorithms

13 Fitting

For fitting in mantid, you can generate a history from a fit in MantidPlot. All the fitting is done using the Fit algorithm. http://docs.mantidproject.org/nightly/algorithms/Fit-v1.html The help page explains how to set up a fitting function, including, ties and constraints.

Exercise 9: Load the GEM63437_focussed.nxs file. Note that the output is a workspace group. get the handle to the second entry. Fit with three Lorentzians plus a linear background. All the widths have to be the same, so tie widths of the second and third Lorentzians to the first. Get the handle to the parameters workspace. You can get the column/row count with the `columnCount()/rowCount()`* functions. To read an individuall cell, use the *`cell(row, col)`* function. Print the ratio of the peak center positions (center2/center1 and center3/center1)*

In [89]: