Hands on Session 1

3) What does this do? i=3 i = i + 1

In [30]:
```
i=3; i=i+1
print(i)
```

```
4
```

This defines i to be 3, the semi-colon is the same as putting a new line and allows you to write several short expressions on the same line. We then redefine the value of i to be the previous value of i plus 1. Note we can also use the c++ style shorthand syntax i += 1

In [31]:
```
i=3; i += 1
print(i)
```

```
4
```

4) Write a function to add two numbers and always return a float

In [32]:
```
def add(a, b):
    return float(a+b)
out = add(3, 4)
print(out)
```

```
7.0
```

5) Use an if statement to print the larger of a or b

In [33]:
```
a = 6.0; b = 4.0
if a>b:
    print(a)
else:
    print(b)
```

```
6.0
```

6) Define a function to raise a floating point number to an integer power N. What changes would you need to make to raise to non-integer powers?

In [34]:
```
def power(a, N):
    return a**N

print("Power of 2 ", power(3., 2))
#Same function works for non-integer power
#because of duck typing
print("Power of 2.5: ",power(3., 2.5))
```

```
('Power of 2 ', 9.0)
('Power of 2.5: ', 15.588457268119896)
```

More Advanced

1) Write a function which combines both 4) and 6) above to get the hypotenuse of a triangle from two side lenghts h2 = o2 + a2

In [35]:
```
def pythag(o, a):
    return power(add(power(o,2),power(a,2)),0.5)

print("Try Pythagorean triple 3, 4, ", pythag(3., 4.))
```

```
('Try Pythagorean triple 3, 4, ', 5.0)
```

2) What does the add_fn do?

In [36]:
```
def add_fn(a, b, fn):
    return fn(a) + fn(b)
```

The final argument is a function. We can pass the function itself in python to other functions (we only call it when we use brackets). In this example, fn is any function which accepts a single argument, so we can use it as follows:

In [37]:
```
def add_fn(a, b, fn):
    return fn(a) + fn(b)

#Convert two ints to float and add
print(add_fn(3, 4, float))
#Convert integers to strings and add together
print(add_fn(3, 4, str))
#Define square function and add squares of numnbers
def square(a):
    return a**2
print(add_fn(3, 4, square))
```

```
7.0
34
25
```

3) Write a recursive factorial function (Note, if you don't know what recursion is, don't worry about this).

In [38]:
```
def factorial( i ):
   if i <1:
       return 1
   else:
       out = i * factorial(i-1)
       return out

factorial(4)
```

Out[38]:
```
24
```

Hand on Session 2

1) Build a sentence s by defining and adding the 4 strings ”is” ,”a”, ”this” and ”sentence” in the right order. Capitalise the first letter of each of the words. Print the first letter of each word. (note no unique way to do these).

In [39]:
```
s="this" + " is" + " a" + " sentence"
t=s.title()

# Split sentence into list of words,
# use an iterator to go through
# and take first letter of each
for i in t.split(" "):
    print(i[0])
```

```
T
I
A
S
```

2) Write a loop to print 10 strings with names: "filename0", "filename1", … "filename9" (note str(i) converts an int to a string)

In [40]:
```
for i in range(10):
    print("filename"+str(i))
```

```
filename0
filename1
filename2
filename3
filename4
filename5
filename6
filename7
filename8
filename9
```

3) Define two lists, one for odd and one for even numbers less than 10. Combine them to form a list of all numbers in the order [1,2,3,4,5,6,7,8,9].

```
In [41]: odd = [2,4,6,8]
         even = [1,3,5,7,9]
         #One option, not very elegant but it works.
         #Take first even element and then loop over
         #remaining lists taking numbers and adding
         #(we need to put brackets
         #on elements to convert to list here)
         combined = [even[0]]
         for i in range(4):
             combined = combined + [odd[i]] + [even[i+1]]
         print(combined)

         # In Python there is an emphasis on simplicity,
         # readability and beauty, see "The Zen of Python".
         # Maybe a clearer way to do this is
         combined = odd+even
         combined.sort()
         print(combined)
```

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

4) Using keys ”even”, ”odd” and ”combined” put lists from 3) in a single dictonary.

```
In [42]: d = {"even":even, "odd":odd}
         d['combined'] = d['even'] + d['odd']
         d['combined'].sort()
         print(d)
```

```
{'even': [1, 3, 5, 7, 9], 'odd': [2, 4, 6, 8], 'combined': [1, 2, 3, 4, 5, 6, 7, 8, 9]}
```

5) Using l = [1,2,3], write a loop to add a number to all elements giving [2,3,4]. Write a function to take in a list l and number N, which adds N to all elements of l.

```
In [43]: l = [1, 2, 3]
         for i in range(len(l)):
             l[i] = l[i] + 1
         print(l)

         def add_to_list(l, N):
             "Function to add one to list"
             for i in range(len(l)):
                 l[i] = l[i] + 1
             return l

         l = [1, 2, 3]
         add_to_list(l, 1.)
```

```
         [2, 3, 4]
Out[43]: [2, 3, 4]
```

More advanced

1) For the string s=”test” and the list l = [”t”,”e”,”s”,”t”], we see s[0] == l[0], s[1] == l[1]. Are they equal? Can you convert the string to a list? What about list to string?

In [44]:
```
s = "test"
l = ["t", "e", "s", "t"]
#Test equality in question
print("s[0] = l[0] ",s[0] == l[0])
print("s[1] = l[1] ",s[1] == l[1])
#But not the same thing
print("But s != l ", s == l)
#Convert string to list
print(list(s))
#List to string, few options here
ls = ""
for t in l:
    ls = ls + t
print(ls)
#Stackoverflow recommended way
print("".join(l))
```

```
('s[0] = l[0] ', True)
('s[1] = l[1] ', True)
('But s != l ', False)
['t', 'e', 's', 't']
test
test
```

2) Define a = [1,2,3]; b = a; b.append(4). Why does a = [1,2,3,4]? what about if you use b = b + [4] instead of append?

In [45]:
```
a = [1,2,3]; b=a; b.append(4)
print(a)
```

```
[1, 2, 3, 4]
```

This is a feature of Python, assigning variable b=a is essentially like giving another name or identifier to [1,2,3] (called a reference). When we append the number 4 to b, we are intriducing the number 4 to the list [1,2,3] which both a and b refer to. If you actually need to create a copy, there is the python copy module,

In [46]:
```
import copy
a = [1,2,3]
b = copy.copy(a)
b.append(4)
print(a, b)
```

```
([1, 2, 3], [1, 2, 3, 4])
```

So a and b now point to seperate lists. There is also a numpy copy for similar reasons. The syntax b = b + [4] defines b = [1,2,3,4] so that variable *a* refers to the list [1,2,3] and b refers to the list [1,2,3,4]

In [47]:
```
a = [1,2,3]; b=a; b=b+[4]
print(a)
```

```
[1, 2, 3]
```

See http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html, especially the section on 'Other languages have "variables"'

Hands on Session 3

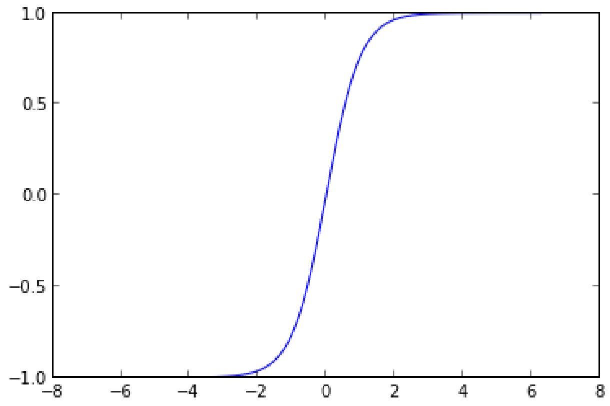
2) Setup a 3 by 3 identity matrix I (ones on the diagonal, zeros off diagonal). Create a 3 by 3 array of random numbers r. Check np.dot(I,r) is as expected

In [48]:
```
import numpy as np
I = np.eye(3,3)
r = np.random.random([3,3]) #Note expects a list of the array dimensions
rdI = np.dot(I,r)
print('Check r == rdI', r == rdI)
```

```
('Check r == rdI', array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]], dtype=bool))
```

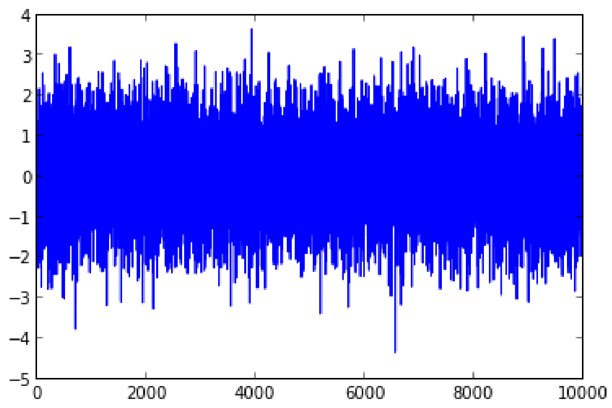
3) Plot a tanh function in the range -2 p to 2 pi using linspace and matplotlib plot.

In [49]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-2*np.pi, 2*np.pi, 1000)
y = np.tanh(x)
plt.plot(x,y)
plt.show()
```



4) Create a 1D array of 10,000 normally distributed random numbers t. Plot as a time history and zoom in to see the detail.

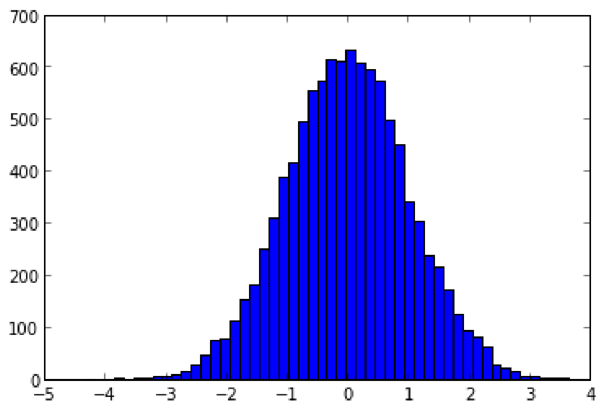
In [50]:
```
import matplotlib.pyplot as plt
import numpy as np

t = np.random.randn(10000)
plt.plot(t)
plt.show()
```



You should get a pop-up window from python/ipython (but not from an inline notebook) which has a toolbar allowing zoom, pan, etc

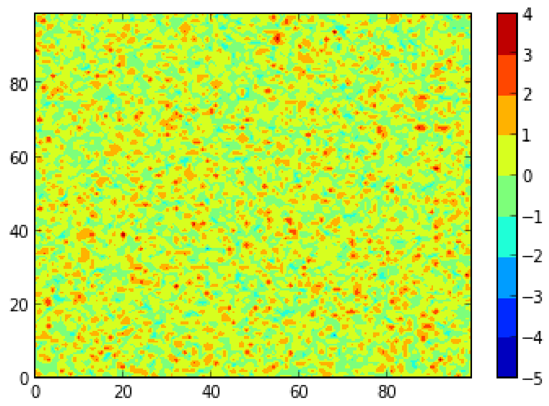
5) Plot a histrogram of the array t from question 4) with 50 bins.

In [51]:
```
plt.hist(t,50)
plt.show()
```



6) Convert array t to a 2D array using t.reshape(100,100) and plot using contourf.

In [52]:
```
tr = t.reshape(100,100)
plt.contourf(tr)
plt.colorbar()
plt.show()
```



7) Create a comma seperated variable file (e.g. 2 columns in excel). Import into Python using np.genfromtxt("./file.csv", delimiter=',') and plot. Check against the plot from excel or other software.

Assuming a file, "file.csv" exists in the correct format, e.g.

x, y

1.0, 1.0

2.0, 4.0

3.0, 9.0

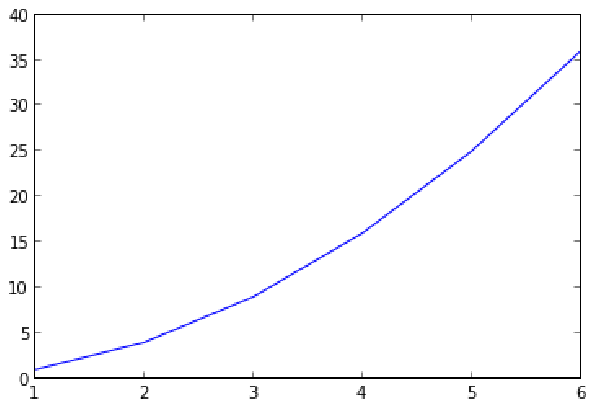
4.0, 16.0

5.0, 25.0

6.0, 36.0

We plot with,

In [53]:
```
data = np.genfromtxt("./file.csv", delimiter=',')
x = data[:,0]
y = data[:,1]
plt.plot(x,y)
plt.show()
```



Please feel free to bring any questions about plotting your own data to the next session.