Functions are basically simplifications in written code.

Some of the functions are intuitive and come pre-installed:

```
In [2]: import numpy as np
```

```
In [3]: np.sqrt(2)   # built in function

Out[3]: 1.4142135623730951
```

```
In [4]: #sqrt and all alike are functions!
```

But what makes functions interesting is that you can define your own !

```
In [5]: def cube(x):
            x=x**3
            return x
```

```
In [6]: cube(4)

Out[6]: 64
```

```
In [7]: if cube(3)>50:
            print 'yes'
        else:
            print 'no'

        no
```

```
In [8]: def sumCubes(x,y):
            a=cube(x)+cube(y)
            return a
```

```
In [9]: sumCubes(2,3)

Out[9]: 35
```

```
In [10]: a  #calling a will return an error since a is a loval variable

         ---------------------------------------------------------------------------
         NameError                                 Traceback (most recent call last)
         <ipython-input-10-ce39b4a9d3ba> in <module>()
         ----> 1 a  #calling a will return an error since a is a loval variable

         NameError: name 'a' is not defined
```

```
In [11]: b=sumCubes(2,3)  # it is possible the assigne the return of the function
                          # to a variable, to a global variable
```

```
In [12]: b

Out[12]: 35
```

```
In [13]: whos

         Variable   Type        Data/Info
         --------------------------------
         b          int         35
         cube       function    <function cube at 0xa512d84>
         np         module      <module 'numpy' from '/us<...>ages/numpy/__init__.pyc'>
         sumCubes   function    <function sumCubes at 0xa512b1c>
```

```
In [14]: # Local variables vs global variables
```

```
a in sumCubes function is a local variable, with "whos" it does not come out.
They can be concidered as boxed (incapsulated) inside the function.
```

```
In [13]:
```

```
In [15]: #
         #also: we can use global variables to go inside the function and then by
         #simply changing the variable
         #we can change thet return of the function
         #
```

```
In [17]: d=3  #first assign something to a variable
```

```
In [19]: def fi():
             s=d**2      # use that variable inside the function!
             return s
```

```
In [21]: fi()   # call the function !

Out[21]: 9
```

```
In [24]: d=4    #change variable value
```

```
In [25]: fi()   # call the function

Out[25]: 16
```

```
Anonymous function aka lambda function
```

```
In [32]: cubew=lambda a,b:a+b      # it is written in a one line more simple
```

```
In [33]: cubew(1,2)

Out[33]: 3
```

```
In [28]: #example
```

In [29]:

```
whos
```

```
Variable   Type        Data/Info
--------------------------------
b          int         35
cube       function    <function cube at 0xa512d84>
cubew      function    <function <lambda> at 0xa51c924>
d          int         4
fi         function    <function fi at 0xa51c6bc>
np         module      <module 'numpy' from '/us<...>ages/numpy/__init__.pyc'>
sumCubes   function    <function sumCubes at 0xa512b1c>
```

In [34]:

```
import math
```

In [35]:

```
t=lambda x: math.factorial(x)
```

In [36]:

```
for i in range(0,10):
    print t(i)
```

```
1
1
2
6
24
120
720
5040
40320
362880
```

In [ ]:

```

```

In [43]:

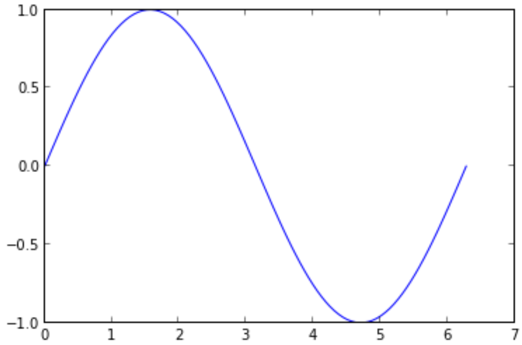
```
#simple example how to ease your work

#once defined the plot desing of linking you can call the function of
#your own plot
```

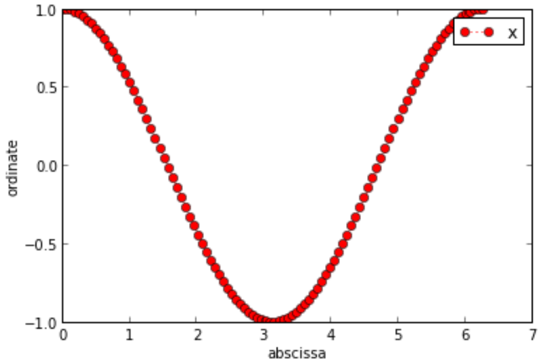
In [38]:

```
%pylab inline
```

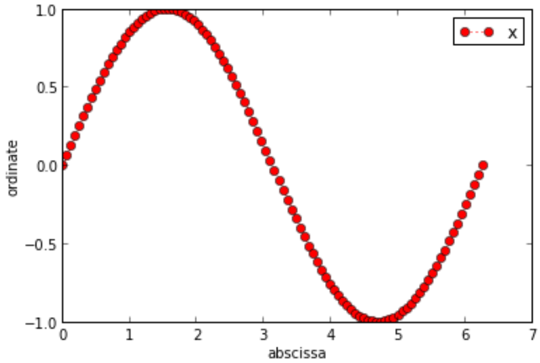
```
Welcome to pylab, a matplotlib-based Python environment [backend:
module://IPython.zmq.pylab.backend_inline].
For more information, type 'help(pylab)'.
```

In [39]:
```
from pylab import *

x=linspace(0,2*pi,100)
f=sin(x)

plot(x,f)             # default style
show()

def myplot(x,y):      # my style
    figure()
    plot(x,y,'ro:')
    xlabel('abscissa')
    ylabel('ordinate')
    legend(['x'])

myplot(x,cos(x))
```

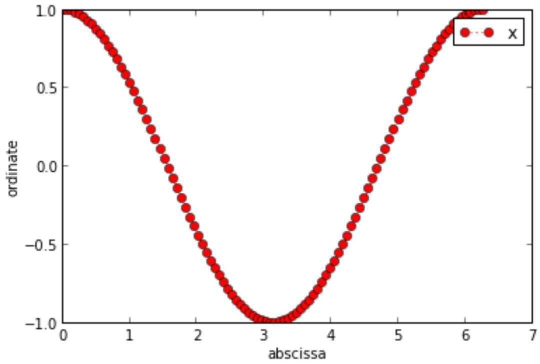




In [40]:
```
myplot(x,f)
```



```
In [41]: myplot(x,cos(x))
```



```
In [ ]:
```