Part 2: Physics Applications:

1- Solving Physics Formulas and math problems
2- Array
3- Vectors
4- Matricx
5- Polynomial
6- Loading Data
7- plotting
8- Fitting
9- Visualization
10- Calculus and Algebra
11- Tables
12- Time
13- Numurical integration and derivative
14- Else you can do with python

1- Solving physics formula and math problems:

for example:
$kE = \frac{1}{2}mv^2$

In [1]:
```
def KE (m,v):
    return 0.5*m*v**2
```

In [2]:
```
KE(10,15)
```

Out[2]:
```
1125.0
```

In [3]:
```
KE = lambda m,v: 0.5 * m * v**2
```

In [4]:
```
KE(10,15)
```

Out[4]:
```
1125.0
```

$$y = 3x^3 - 5x^2 - \frac{3}{4}x + 1$$

In [5]:
```
def y(x):
    res = 3 * x**3 - 5 * x**2 - (3.0/4) * x + 1
    return res
```

In [6]:
```
y(10)
```

Out[6]:
```
2493.5
```

In [7]:
```
y = lambda x: 3 * x**3 - 5 * x**2 - (3.0/4) * x + 1
```

In [8]:
```
y(10)
```

Out[8]:
```
2493.5
```

In [9]:
```
import math
print math.sin(1)
print math.cos(1)
print math.tan(0.5)
print math.radians(90)
print math.degrees(math.pi)
print math.e
print math.exp(10)
print math.log(5) # =>  ln(5)
print math.log10(5)
```

```
0.841470984808
0.540302305868
0.546302489844
1.57079632679
180.0
2.71828182846
22026.4657948
1.60943791243
0.698970004336
```

2- Array:

array هي مجموعة تتميز ب

1- القابلية للعمليات الرياضية مثل الضرب وغيره

2- السرعة في العمليات الحسابية

In [5]:
```
import numpy as np  # np وارمز لها اختصارا ب  numpy اي: استدعي مكتبة
```

In [11]:
```
# Array and math operations
# numpy يجب أولا ان ندخل داخل مكتبة array لكي نستدعي
# وذلك من خلال كتابة اسم المكتبة ثم وضع نقطة
a = np.array([0,1,2,3,4,5,6,7,8,9]) # array هكذا نكتب مجموعة
a
```

Out[11]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [12]:
```
b = np.arange(10) # تبدأ من الصفر وتنتهي الى ما قبل العشرة array يقوم بتكوين مجموعة
#list يشبه التعامل مع القائمة array ملاحظة:التعامل مع مجموعة
b
```

Out[12]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [13]:
```
c = np.linspace(0,10,21)
#21 عنصر يبدأ من الصفر وينتهي عند العشرة وعدد عناصرها array يقوم بإخراج
c
```

Out[13]:
```
array([  0. ,   0.5,   1. ,   1.5,   2. ,   2.5,   3. ,   3.5,   4. ,
         4.5,   5. ,   5.5,   6. ,   6.5,   7. ,   7.5,   8. ,   8.5,
         9. ,   9.5,  10. ])
```

In [14]:
```
a * 2
#list لاحظ ان كل عنصر تم ضربه ب 2، وهذا لا يمكن أن يحصل مع القائمة
```

Out[14]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [15]:
```
a ** 2
```

Out[15]:
```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

In [16]:
```
a + 1
```

Out[16]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [17]:
```
KE(5,a) #list لاحظ أن كل عنصر في a دخل في الدالة KE وهذا لا يحصل مع القائمة
```

Out[17]:
```
array([   0. ,    2.5,   10. ,   22.5,   40. ,   62.5,   90. ,  122.5,
        160. ,  202.5])
```

In [18]:
```
b = np.array([1,2,3])
c = np.array([0,0,2])
b*c
```

Out[18]:
```
array([0, 0, 6])
```

In [19]:
```
2 * b + c
```

Out[19]:
```
array([2, 4, 8])
```

3- Vectors المتجهات

In [20]:
```
# يمكن استخدام array في ضرب المتجهات
D = np.array([1,2,3])
E = np.array([2,4,0])
print 'D = 1i + 2j + 3k'
print 'E = 2i + 4j + 0k'
dot = np.dot(D,E) # ضرب قياسي dot product
print 'D . E = ', dot
cross = np.cross(D,E) # ضرب اتجاهي cross product
print 'D X E = {}i + {}j + {}k'.format(*cross)
```

```
D = 1i + 2j + 3k
E = 2i + 4j + 0k
D . E =  10
D X E = -12i + 6j + 0k
```

4- Matrix المصفوفات

In [21]:
```
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
A
```

Out[21]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [22]:
```
A * 2
```

Out[22]:
```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

In [23]:
```
A[0] # <-- اختصار/ A[0,:]
```

Out[23]:
```
array([1, 2, 3])
```

In [24]:
```
A[1]
```

Out[24]:
```
array([4, 5, 6])
```

In [25]:
```
A[0,0]
```

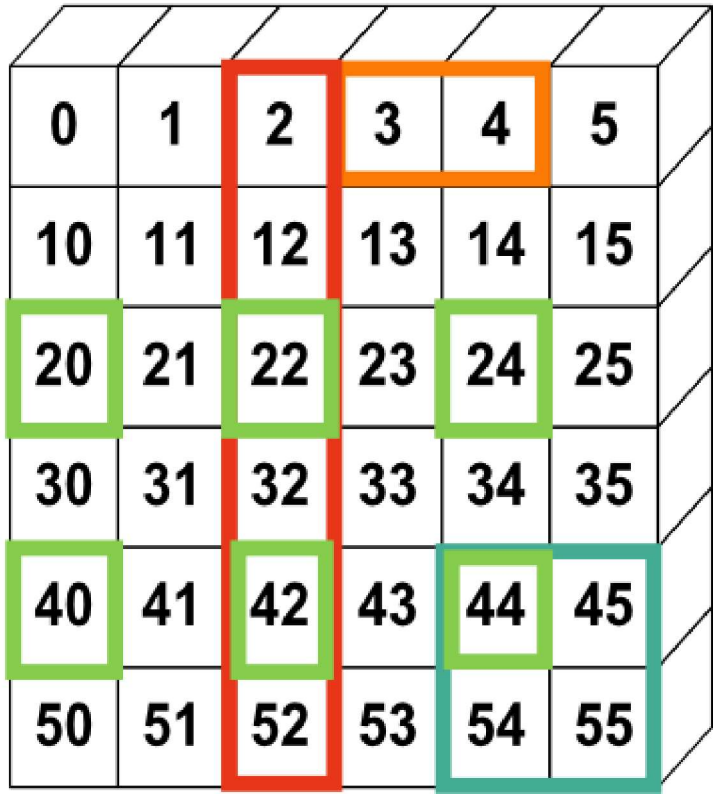
Out[25]:
```
1
```

In [26]:
```
A[:,1]
```

Out[26]:
```
array([2, 5, 8])
```

لكي تتعامل مع عناصر المصفوفة عليك أن تفهم أن
A[رقم العمود ، رقم الصف]
A[النهاية من الأعمدة : البداية من الأعمدة , النهاية من الصفوف :البداية من الصفوف]
الصورة التالية توضح الطريقة

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



In [27]:
```
print A.shape #    يعطيك (عدد الأعمدة ، عدد الصفوف)
print A.size # عدد العناصر
print A.max()
print A.min()
print A.sum() # مجموع قيم العناصر
print A.mean() # المتوسط
```

```
(3L, 3L)
9
9
1
45
5.0
```

In [28]:
```
B = np.array([[2,4],
              [1,3]])
C = np.array([[0,1],
              [1,2]])

B * C #  إحذر! هذا ضرب مباشر وليس ظرب مصفوفات
```

Out[28]:
```
array([[0, 4],
       [1, 6]])
```

In [29]:
```
np.dot(B,C) # هذا ضرب صحيح للمصفوفات
```

Out[29]:
```
array([[ 4, 10],
       [ 3,  7]])
```

In [30]:
```
F = np.array([[1,0,2],
              [3,2,4],
              [1,5,8]])
print np.linalg.det(F) # طريقة ايجاد المحددة للمصفوفة
print
print F.T # طريقة قلب المصفوفة بحيث تتحول الصفوف الى اعمدة
print
print np.linalg.inv(F) # F^-1 مقلوب المصفوفة
```

```
22.0

[[1 3 1]
 [0 2 5]
 [2 4 8]]

[[-0.18181818  0.45454545 -0.18181818]
 [-0.90909091  0.27272727  0.09090909]
 [ 0.59090909 -0.22727273  0.09090909]]
```

5- Polynomial كثيرة الحدود

يمكن من خلال مكتبة numpy استخدام poly1d والذي يقوم بصناعة كثيرة حدود
حيث أنه يطلب منك فقط أن تقوم بإدخال المعاملات، وهو يقوم بالباقي. مثال:

$p(x) = 2x^2 - 5x + 3$

نلاحظ أن كثيرة الحدود هذه من الدرجة الثانية لأن اكبر اس فيها هو 2
وانها تحتوي على 3 معاملات

In [31]:
```
p = np.poly1d([2,-5,3])
print 'p(0) = ', p(0) #  يقوم بتعويض x بصفر ثم يعطيك الناتج
print 'p(1) = ', p(1)
print 'p(10) = ', p(10)
```

```
p(0) =  3
p(1) =  0
p(10) =  153
```

6- Loading Data تحميل البيانات

يمكنك تحميل بيانات من ملف خارجي والاستفادة منها

In [32]:
```
data = np.loadtxt('C:/Users/User/Desktop/data.txt')
data # لاحظ انه يأخذ البيانات ثم يفصل المحاور عن بعضها
```

Out[32]:
```
array([[  1.,   2.],
       [  2.,   4.],
       [  3.,   5.],
       [  4.,   8.],
       [  5.,  10.],
       [  6.,  14.],
       [  7.,  20.]])
```

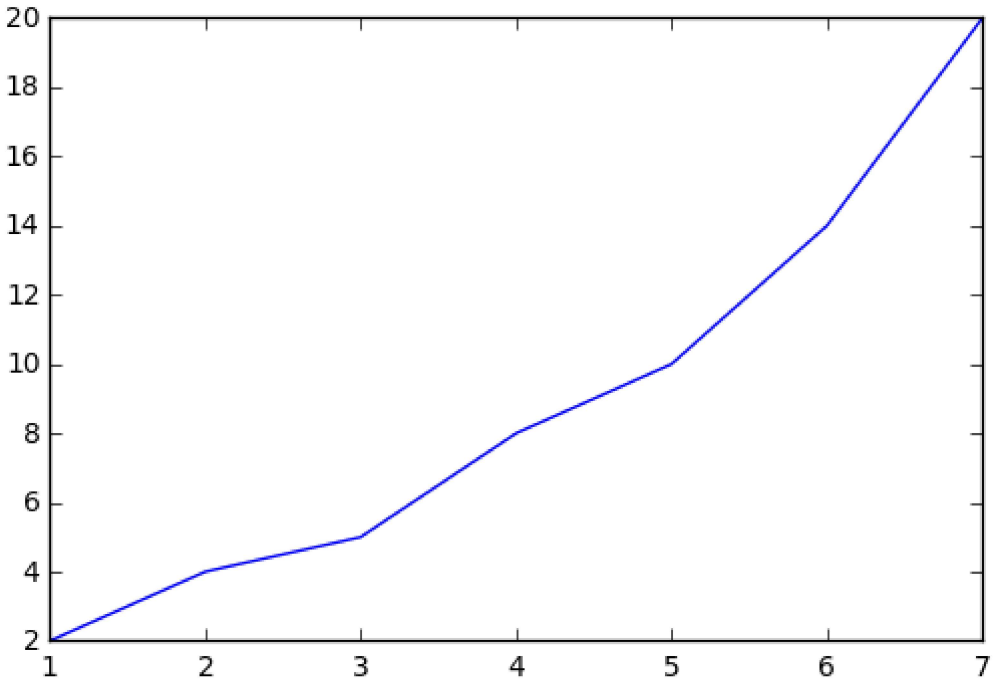
In [33]:
```
x = data[:,0] # يعطيك محور x
y = data[:,1] # يعطيك محور y
print x
print y
```

```
[ 1.  2.  3.  4.  5.  6.  7.]
[  2.   4.   5.   8.  10.  14.  20.]
```

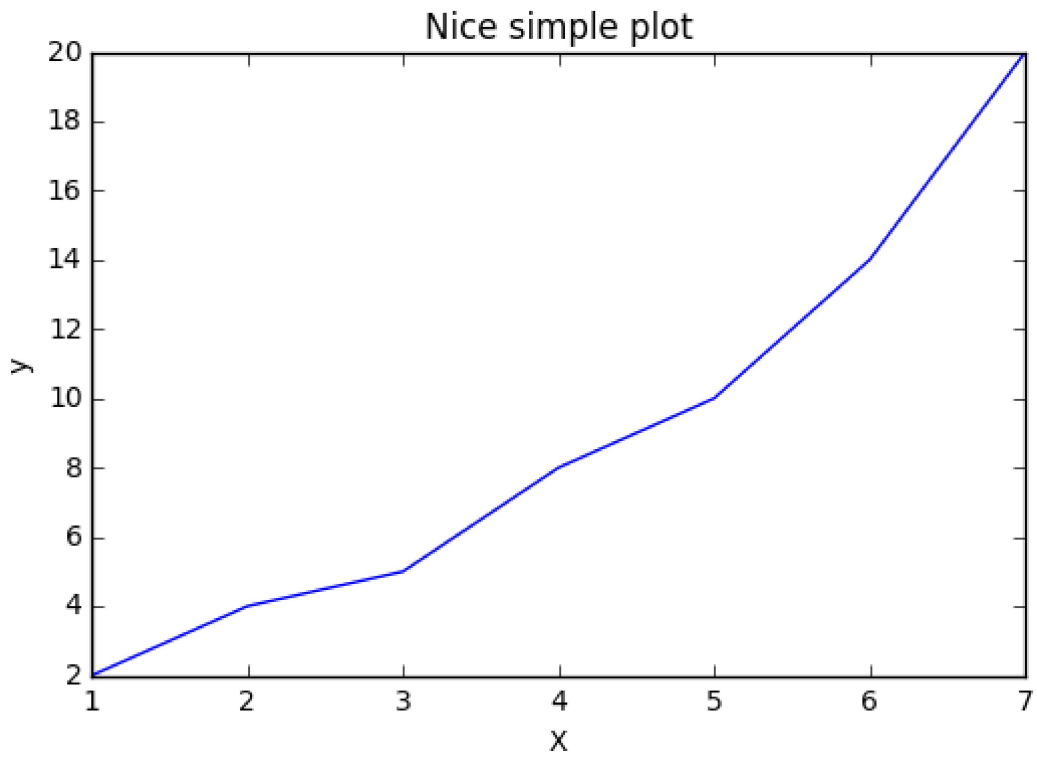
7- Plotting الرسم البياني

In [6]:
```
import matplotlib.pyplot as pl # هذه هي المكتبة التي نستخدمها في الرسم البياني
%matplotlib inline
# يقوم هذا الكود بإخراج الرسم البياني في نفس الصفحة
```

In [35]:
```
# تستطيع أن ترسم رسم بياني من خلال سطر واحد فقط!!!
pl.plot(x,y) #أي أرسم x على محور السينات و y على محور الصادات
pl.show() # أي أعرض الرسم البياني
# ملاحظة: اخذنا قيم x,y من الملف السابق
```

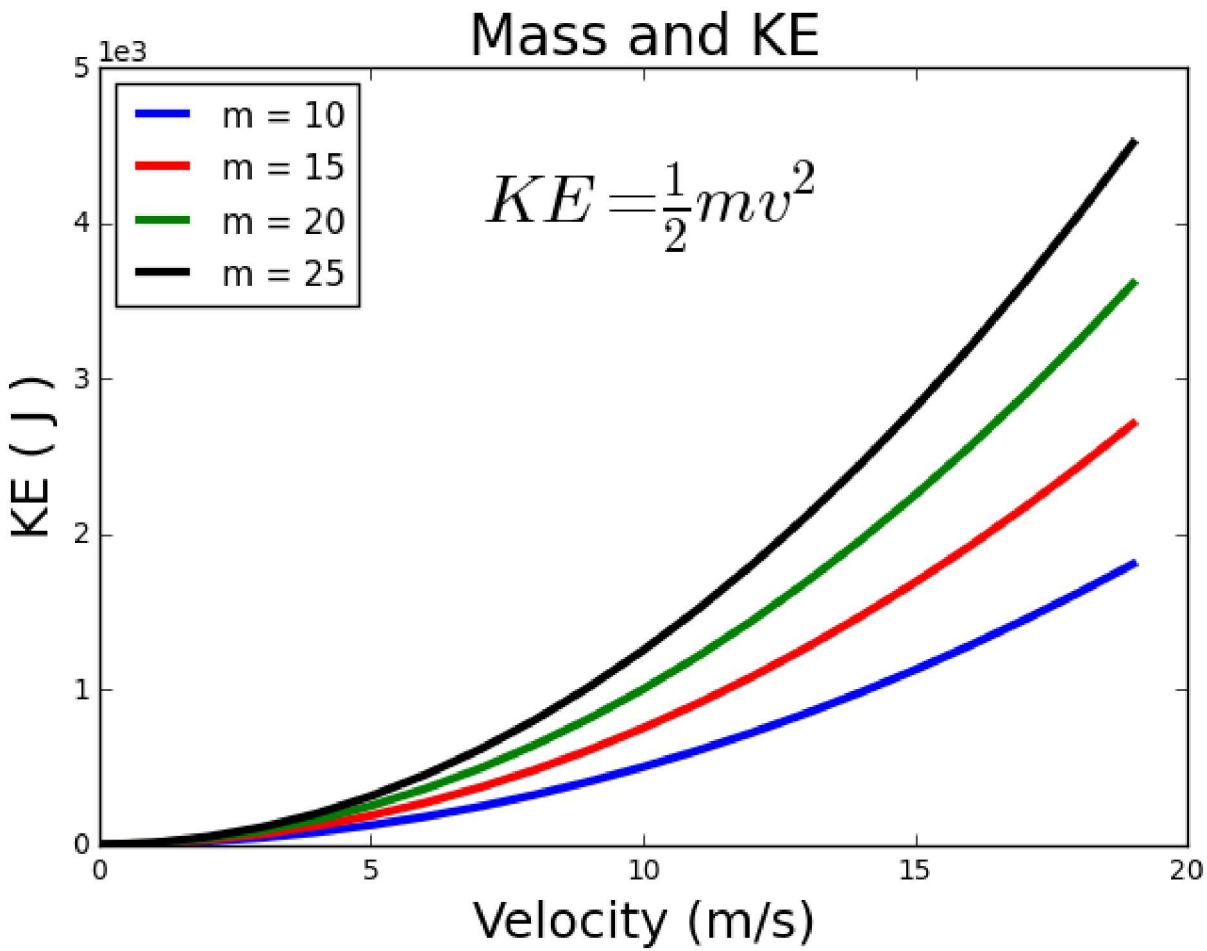


In [36]:
```
pl.plot(x,y)
pl.title('Nice simple plot') # عنوان الرسم البياني
pl.xlabel('X') # اسم محور السينات
pl.ylabel('y') # اسم محور الصادات
pl.show()
```

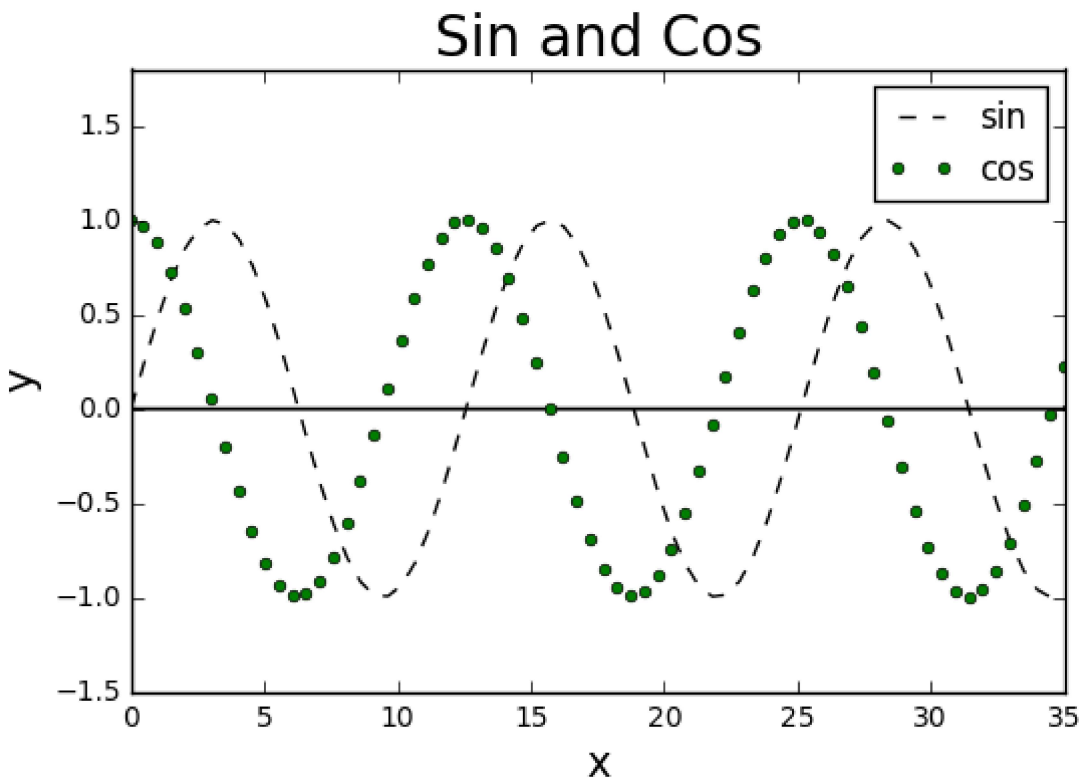


In [37]:

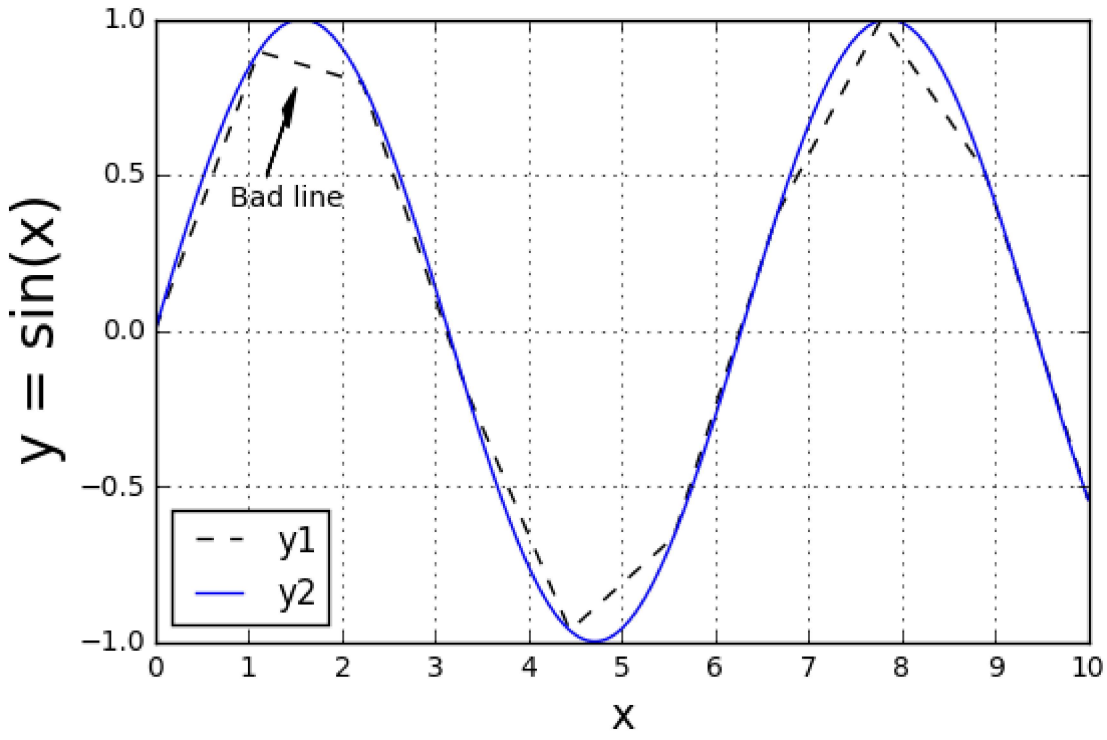
```
KE = lambda m,v: 0.5 * m * v**2
v = np.arange(20)
k1 = KE(10,v) # كتبنا معادلة KE في اول الدروس
k2 = KE(15,v)
k3 = KE(20,v)
k4 = KE(25,v)

pl.figure(figsize=(7, 5), dpi=200) # figsize أبعاد الرسم ، dpi يعطيك جودة الرسم
pl.title('Mass and KE' , fontsize = 20)# عنوان الرسم البياني
pl.plot(v,k1,'b',label ='m = 10',lw = 3)
# 'b' أي لون الخط أزرق ، label اسم الخط ، lw سمك الخط
pl.plot(v,k2,'r', label = 'm = 15',lw = 3) # r خط أحمر
pl.plot(v,k3,'g',label ='m = 20',lw = 3) # g خط أخضر
pl.plot(v,k4,'k', label = 'm = 25', lw = 3) # k خط أسود
pl.legend(loc = 'best') # legend يضع لك المربع التوضيحي، loc أي موقع المربع، 'best' أي افضل مكان
pl.xlabel('Velocity (m/s)',fontsize = 18)
pl.ylabel('KE ( J )',fontsize = 18)
pl.text(7,4000,'$KE = \\frac{1}{2}mv^2$',fontsize = 25) # كتابة نص داخل الرسم البياني
#text(x,y,"النص", fontsize = حجم الخط)
pl.ticklabel_format(axis='y', scilimits=(0,0)) # يختصر الارقام على محور y الى 10*1 أس رقم مناسب
pl.savefig('C:/Users/User/Desktop/KE_vs_V.png',dpi = 200)
# يقوم بحفظ الرسم البياني في المكان الذي تريده في جهازك
pl.show()
```


Mass and KE
1e3
m = 10
m = 15
m = 20
m = 25
$KE = \frac{1}{2}mv^2$
KE (J)
Velocity (m/s)
0
1
2
3
4
5
0
5
10
15
20

In [38]:
```
x = np.linspace(0,35,70)
y1 = np.sin(0.5*x)
y2 = np.cos(0.5*x)
pl.title('Sin and Cos',fontsize = 20)
pl.plot(x,y1,'k--',label ='sin')#  خط اسود مقطع'k--'
pl.plot(x,y2,'go', label = 'cos',markersize=4)# حجم الدوائر markersize ، دوائر خضراء'go'
pl.legend(loc = 'best')
pl.ylim(-1.5,1.8) # 1.8،-1.5 أي تقسيم محور y يكون بين
#pl.xlim(x_i , x_f) يوجد أيضا
pl.xlabel('x',fontsize = 15)
pl.ylabel('y',fontsize = 15)
pl.arrow(0,0,35,0)# يقوم برسم خط مستقيم
# arrow(x نقطة بداية الخط لمحور,y نقطة بداية الخط لمحور x, مقدار الزيادة في محور y, مقدار الزيادة في محور)
pl.savefig('C:/Users/User/Desktop/sin_cos.png',dpi = 200)
pl.show()
```



In [39]:
```
x1 = np.linspace(0,10,10) #  لاحظ أن النقاط هنا قليلة بالتالي ستكون الدقة ضعيفة
x2 = np.linspace(0,10,1000) # لاحظ ان النقاط هنا كثيرة بالتالي سيكون الرسم افضل
y1 = np.sin(x1)
y2 = np.sin(x2)
pl.plot(x1,y1,'k--',label = 'y1' )
pl.plot(x2,y2,'b' ,label = 'y2' )
pl.legend(loc = 0) # صفر تعني أحسن مكان
pl.grid() # يقوم بوضع مربعات او شبكة في الرسم البياني
pl.arrow(1.2,0.5,0.2,0.18,head_width = 0.08, head_length = 0.15,fc = 'k')
# لون رأس السهم fc ، طول رأس السهم head_length ، عرض رأس السهم head_width
pl.text(0.8,0.4,'Bad line')
pl.xticks(range(11)) # يحدد عدد التقسيمات على محور x
pl.xlabel('x',fontsize = 15)
pl.ylabel('y = sin(x)',fontsize = 20)
pl.show()
```

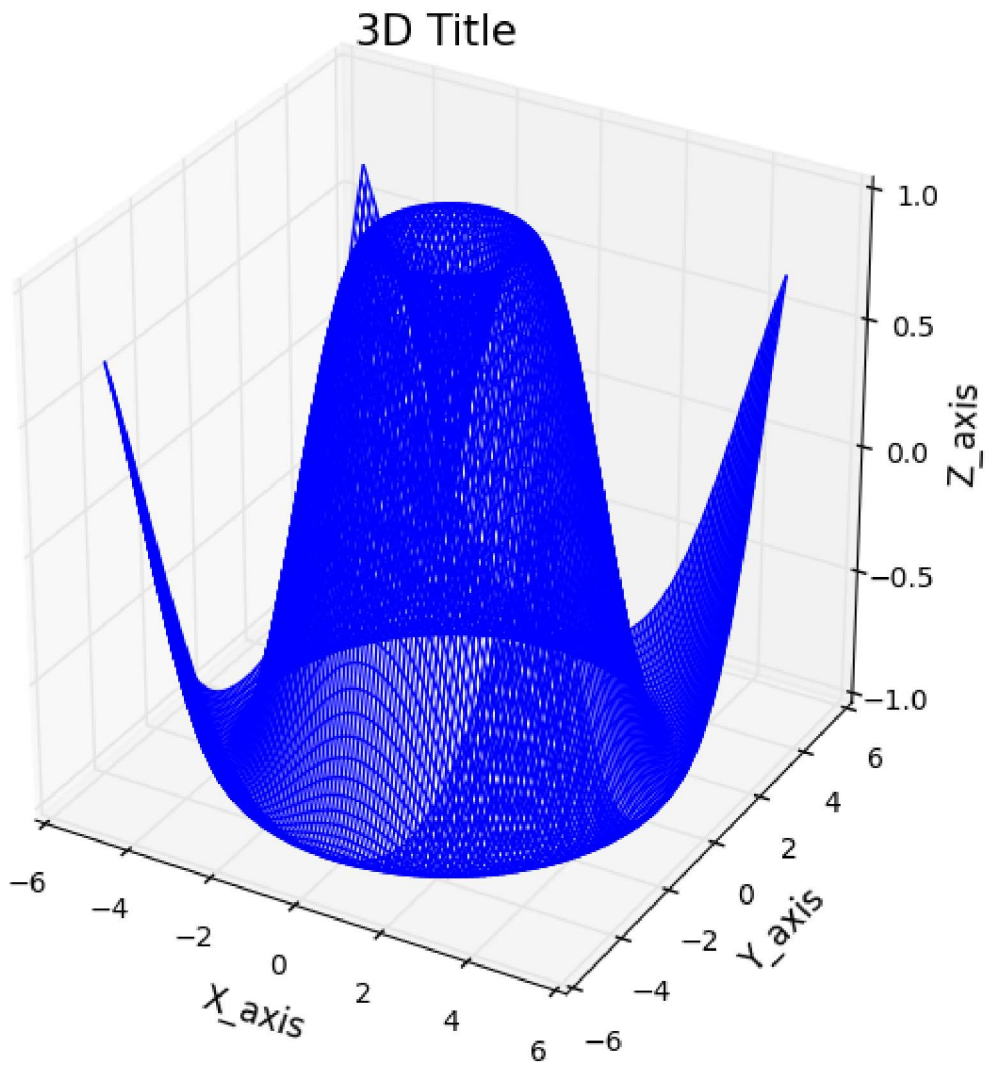


3D Plotting

In [2]:
```
from mpl_toolkits.mplot3d import Axes3D # مكتبة للرسم الثلاثي الابعاد
```

In [56]:
```
x = np.linspace(-5,5,100)
y = np.linspace(-5,5,100)
x, y = np.meshgrid(x, y) #يحول قيم x و y الى بعدين بعد ان كانت بعد واحد
z = np.sin(np.sqrt(x**2 + y**2))
fig = pl.figure(figsize=(7,7)) # يقوم بوضع إطار رسم ابعاده 7 في 7
ax = fig.add_subplot(1,1,1, projection='3d') # يقوم بجعل الرسم البياني ثلاثي الابعاد
ax.plot_wireframe(x, y, z,color = 'b') # يقوم بالرسم
ax.set_title('3D Title',fontsize = 15) # العنوان
ax.set_zlabel('Z_axis',fontsize = 12) # اسم محمور z
ax.set_xlabel('X_axis',fontsize = 12) # اسم محمور x
ax.set_ylabel('Y_axis',fontsize = 12) # اسم محمور y
ax.set_zlim3d(-1, 1)  # حدود محور z
```

Out[56]: (-1, 1)



8- Fitting

curve_fit:

عندما يكون لدينا بيانات لها شكل دالة معينة فإننا نقوم بالاتي لكي نعمل لها fitting

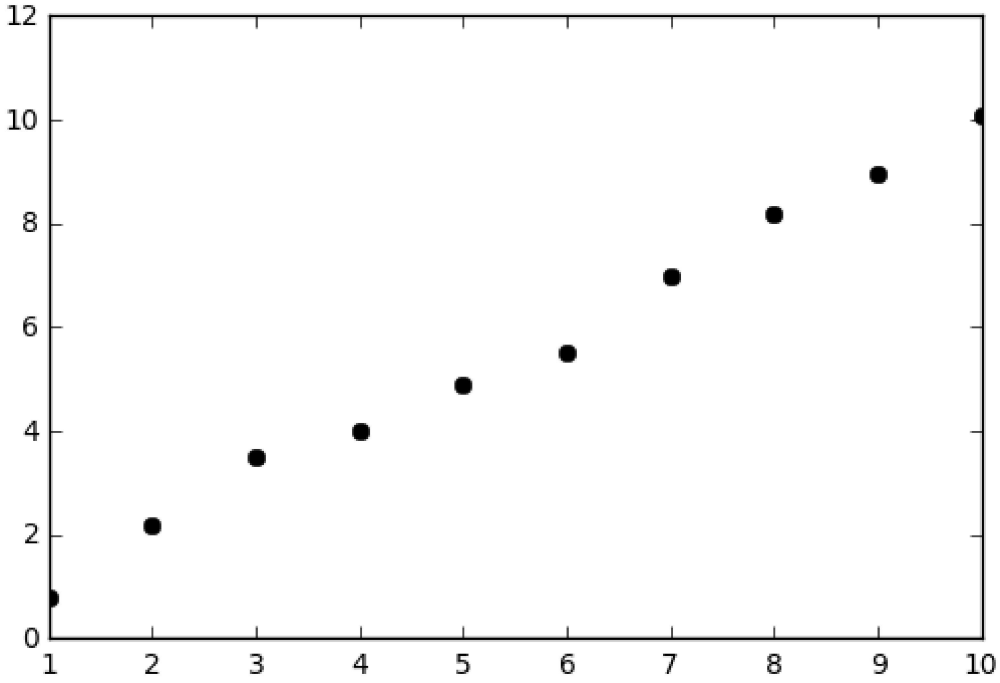
1- نكتب الدالة المناسبة على شكل lambda أو def

2- نستخدم هذا الامر (curve_fit)

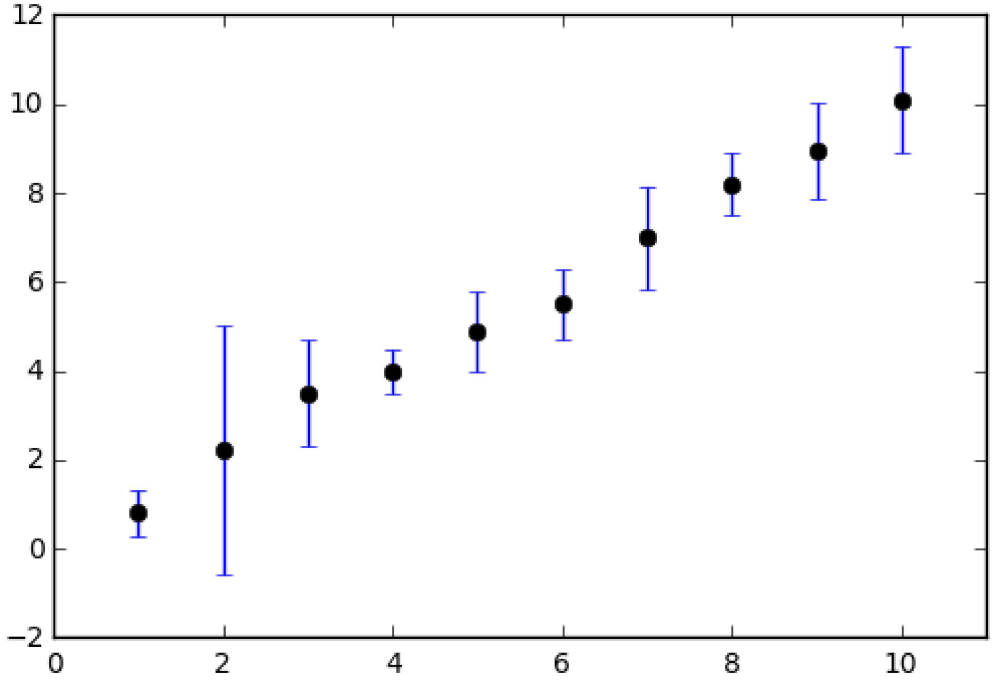
curve_fit (بيانات المحورالصادي, بيانات المحور السيني,الدالة المناسبة)

In [40]:
```
from scipy.optimize import curve_fit
```

In [41]:
```
x = [1,2,3,4,5,6,7,8,9,10]
y = [0.8,2.2,3.5,4,4.9,5.5,7,8.2,8.95,10.1]# قراءات من تجربة ما
pl.plot(x,y,'ko')
pl.show()
```



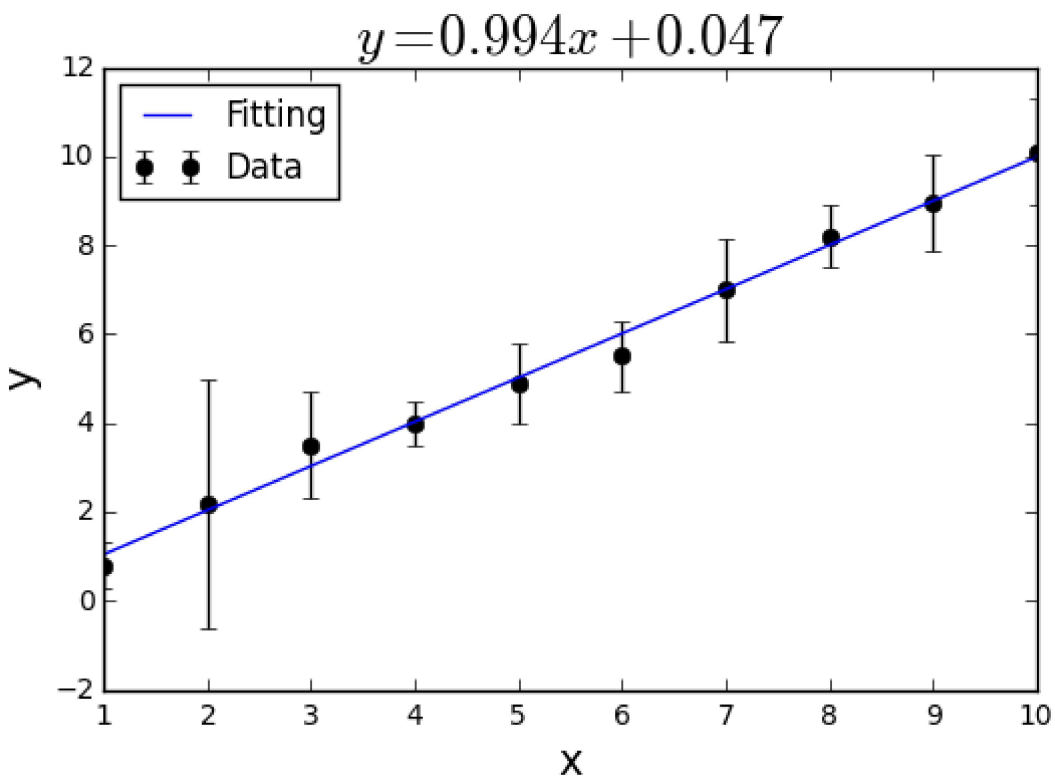
In [42]:
```
# الان لنفترض وجود نسبة خطأ في القراءات
y_err = [0.5,2.8,1.2,0.5,0.9,0.8,1.15,0.7,1.1,1.2] #نسبة الخطأ لكل نقطة في البيانات
# errorbar دالة تقوم بوضع علامة الخطأ في الرسم البياني
pl.errorbar(x,y,y_err,fmt='ko',ecolor='b') #  اي ارسم البيانات شكل نقاط لونها اسود  fmt='ko'
# أي اجعل علامة الخطأ باللون الازرق  ecolor='b'
pl.xlim(0,11)
pl.show()
```



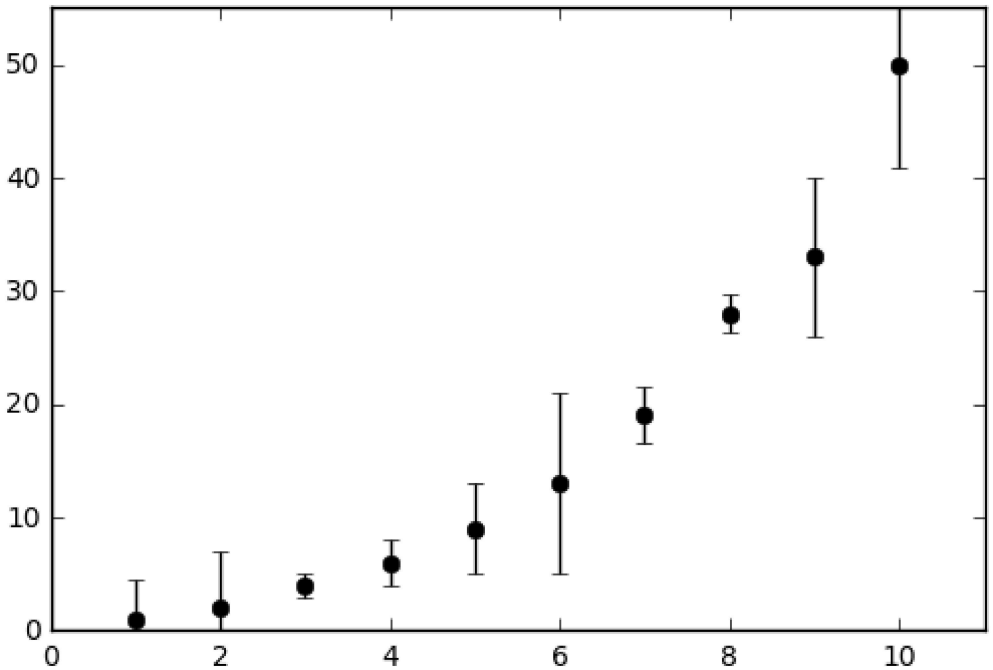
In [43]:
```
# بما أن القراءات أعلاه يمكن تمثيلها بمعادلة الخط المستقيم عندئذ نستخدم المعادلة التالية
# y(x) = a.x + b
# حيث a ثابت يمثل الميل و b ثابت يمثل الجزء المقطوع من محور الصادات
fun = lambda x,a,b : a * x + b
fit,cov = curve_fit(fun,x,y)
# حيث fit يحتوي على افضل قيمة ل a و b
# من خلال cov نستطيع حساب نسبة الخطأ للبيانات
fit
```

Out[43]:
```
array([ 0.99424242,  0.04666667])
```

In [44]:
```
a = fit[0]
b = fit[1]
pl.errorbar(x,y,y_err,fmt = 'ko',label = 'Data')
x = np.array(x)
pl.plot(x,fun(x,a,b),label='Fitting') # fun يجب أن يكون x  هنا عبارة عن array حتى يدخل في
pl.legend(loc = 0)
pl.xlabel('x',fontsize = 15)
pl.ylabel('y',fontsize = 15)
pl.title('$y = {:.3f}x+{:.3f}$'.format(*fit),fontsize = 20)
pl.show()
```



In [45]:
```
x = np.array([1,2,3,4,5,6,7,8,9,10])
y = np.array([1,2,4,6,9,13,19,28,33,50])
y_err = [3.5,5,1,2,4,8,2.5,1.7,7,9.2]

pl.errorbar(x,y,y_err,fmt = 'ko')
pl.xlim(0,11)
pl.ylim(0,55)
pl.show()
```



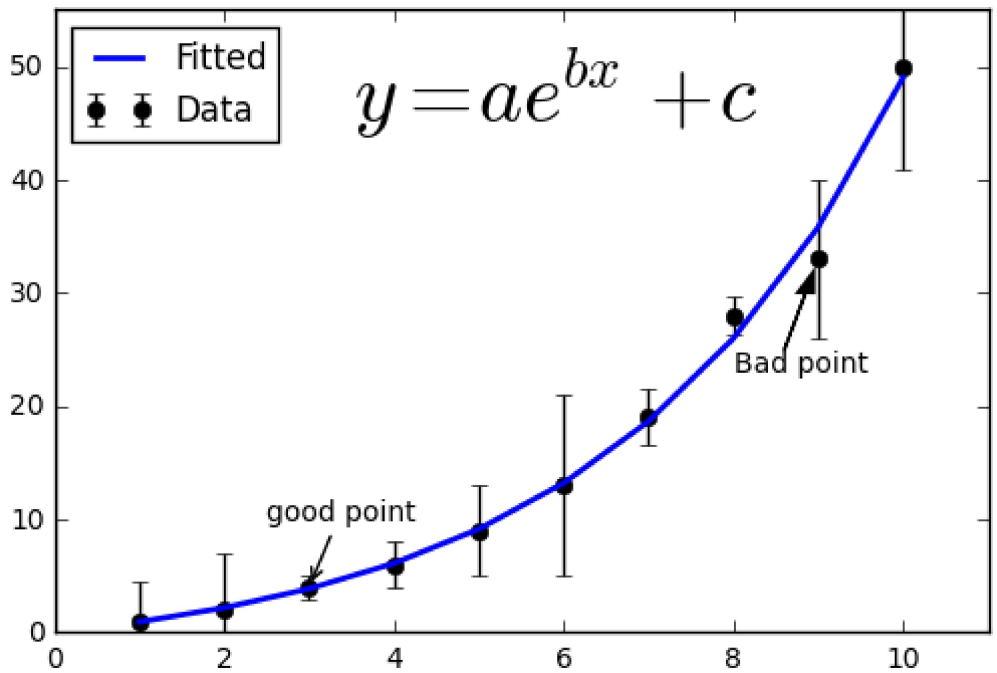
In [46]:
```
# نلاحظ أن البيانات أعلاه تشبه الدالة الاسية، لذلك نكتب الاتي
def fun(x,a,b,c):
    return a*np.exp(b*x)+c  # fun(x,a,b,c) = a . e^x  + c
pfit,pcov = curve_fit(fun,x,y)
pfit
```

Out[46]:
```
array([ 2.70912135,  0.294937  , -2.69126787])
```

In [47]:
```
a,b,c = pfit
print 'a = ', a
print 'b = ', b
print 'c = ', c
```

```
a =  2.7091213501
b =  0.294936995377
c =  -2.69126787259
```

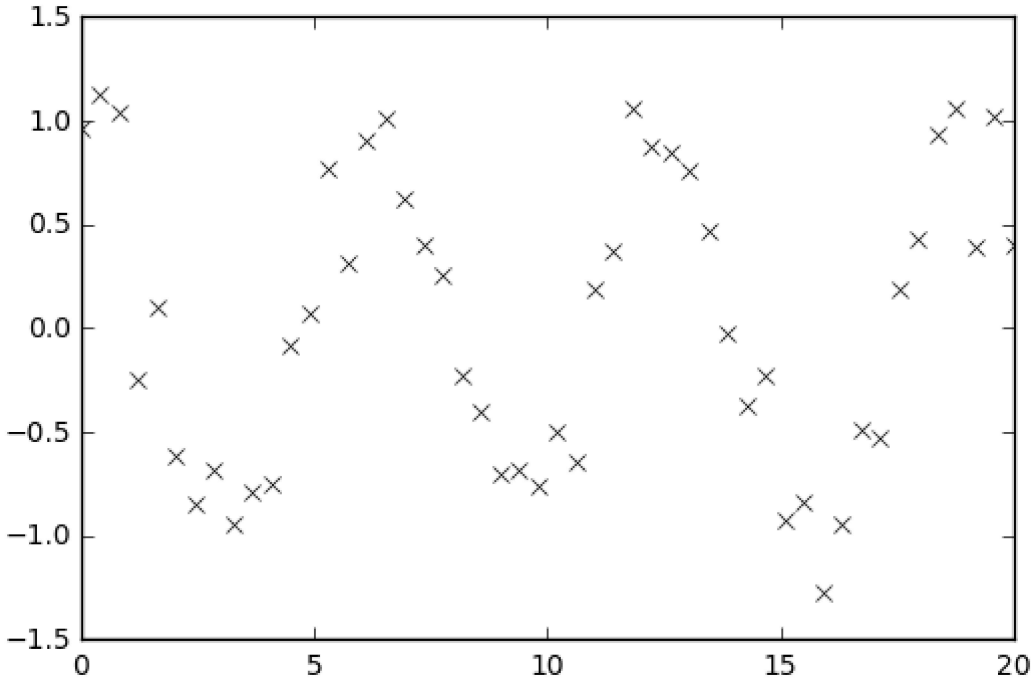
In [48]:
```
pl.errorbar(x,y,y_err,fmt ='ko',label = 'Data')
pl.plot(x,fun(x,a,b,c),'b',label = 'Fitted' , lw = 2)
pl.legend(loc = 0)

pl.text(3.5,45,'$y = a e^{bx} + c$',fontsize = 30)

pl.arrow(8.6,25,0.25,5, head_width = 0.2, head_length = 2 , ec = 'k' , fc = 'k' )
pl.text(8,23,'Bad point')
pl.annotate('good point ',xy = (3,4), xytext = (2.5,10), arrowprops = {'arrowstyle':'->',})
pl.xlim(0,11)
pl.ylim(0,55)

pl.show()
```



In [49]:
```
data = np.loadtxt('C:\Users\User\Desktop\cos_data.txt',float)
```

In [50]:
```
x = data[:,0]
y = data[:,1]
pl.plot(x,y,'kx')
```

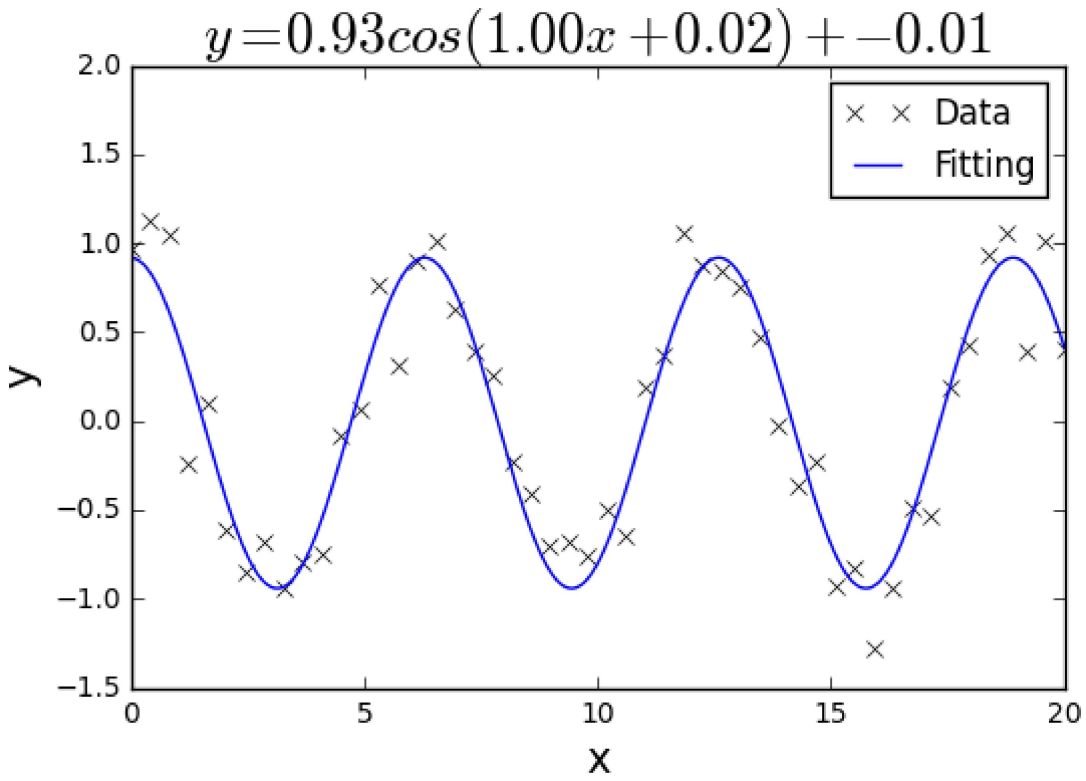
Out[50]: [<matplotlib.lines.Line2D at 0x192abbe0>]



In [51]:
```
# cos نلاحظ أن البيانات اعلاه تشبه دالة
fun = lambda x,a,w,b,c: a * np.cos(w * x + b) + c
pfit,pcov = curve_fit(fun,x,y)
pfit
```

Out[51]: array([0.93019157, 0.99618855, 0.01836   , -0.01202377])

In [52]:
```
pl.plot(x,y,'kx',label = 'Data')
x0 = np.linspace(0,20,1000)
pl.plot(x0,fun(x0,*pfit),'b',label = 'Fitting')
pl.legend(loc = 0)
pl.xlabel('x',fontsize = 15)
pl.ylabel('y',fontsize = 15)
pl.ylim(-1.5,2)
pl.title('$y = {:.2f}cos({:.2f}x+{:.2f})+{:.2f} $'.format(*pfit),fontsize = 20)
pl.show()
```



Polynomial Fitting

اذا كانت نقاط البيانات(من التجربة)التي تريد وصفها (يمكن تمثيلها) على شكل كثيرة حدود مثل

$y = ax + b$

هذه كثيرة حدود من الدرجة الأولى لها معاملين a,b

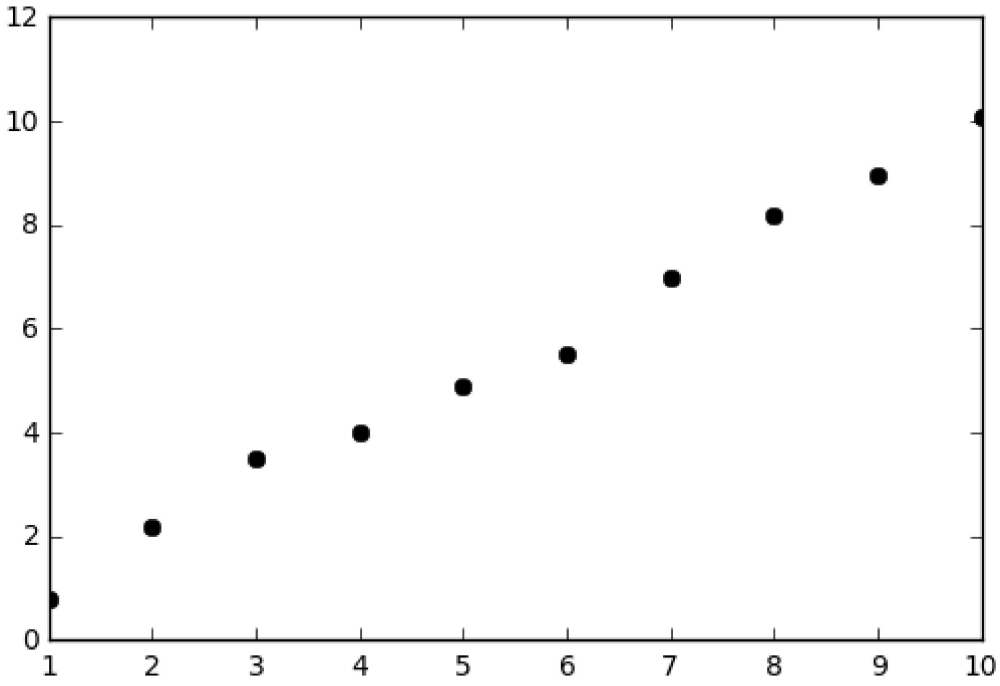
$y = ax^2 + bx + c$

هذه كثيرة حدود من الدرجة الثانية لما ثلاثة معاملات

نستطيع عندئذ اختصار الطريقة أعلاه باستخدام الامر polyfit

polyfit(x,y,درجة كثيرة الحدود)

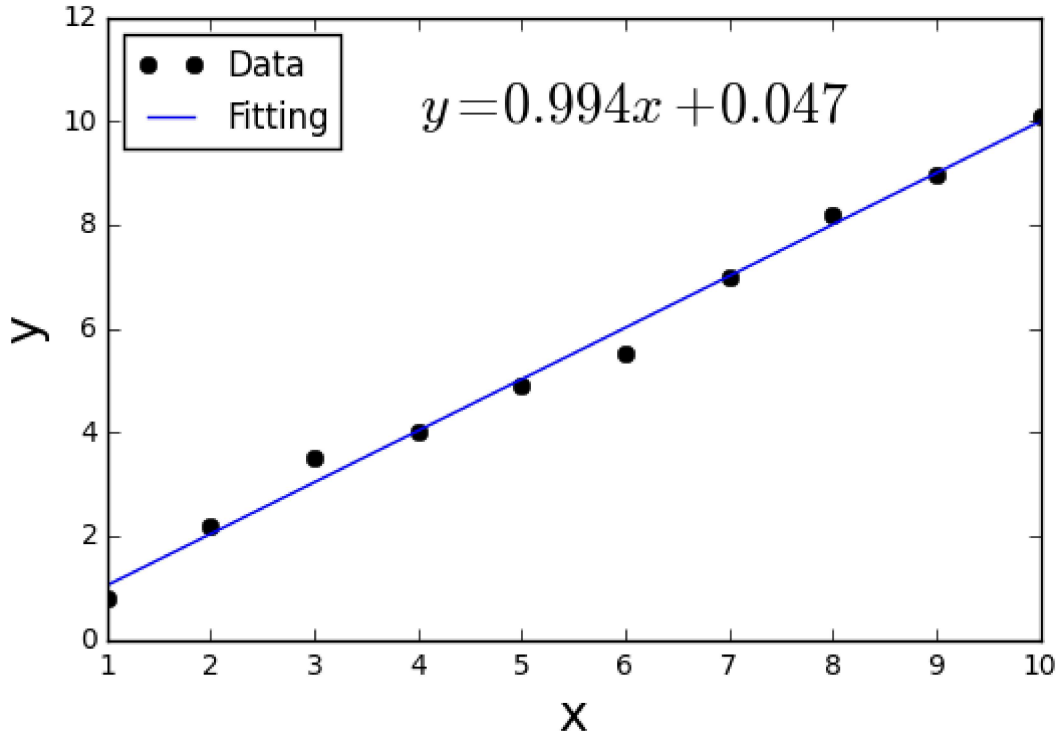
In [53]:
```
x = [1,2,3,4,5,6,7,8,9,10]
y = [0.8,2.2,3.5,4,4.9,5.5,7,8.2,8.95,10.1]# قراءات من تجربة ما
pl.plot(x,y,'ko')
pl.show()
```



In [54]:
```
# الان بما أن البيانات السابقة يمكن تمثيلها على شكل معادلة الخط المستقيم(أي كثيرة حدود
# من الدرجة الأولى) فإننا عندئذ نقوم بالبحث عن المعاملين a و b
z = np.polyfit(x,y,1)# تضع محور x  و محور y وأخيرا درجة كثيرة الحدود
# لاحظ بأن مخرجات polyfit هي المعاملات لكثيرة الحدود
print z # الان تمت عملية fitting
print 'y = {}x + {}'.format(*z) # هذه هي المعادلة النهائية
print 'Slope = ', z[0] # هذا هو الميل
```

```
[ 0.99424242  0.04666667]
y = 0.994242424242x + 0.0466666666667
Slope =  0.994242424242
```

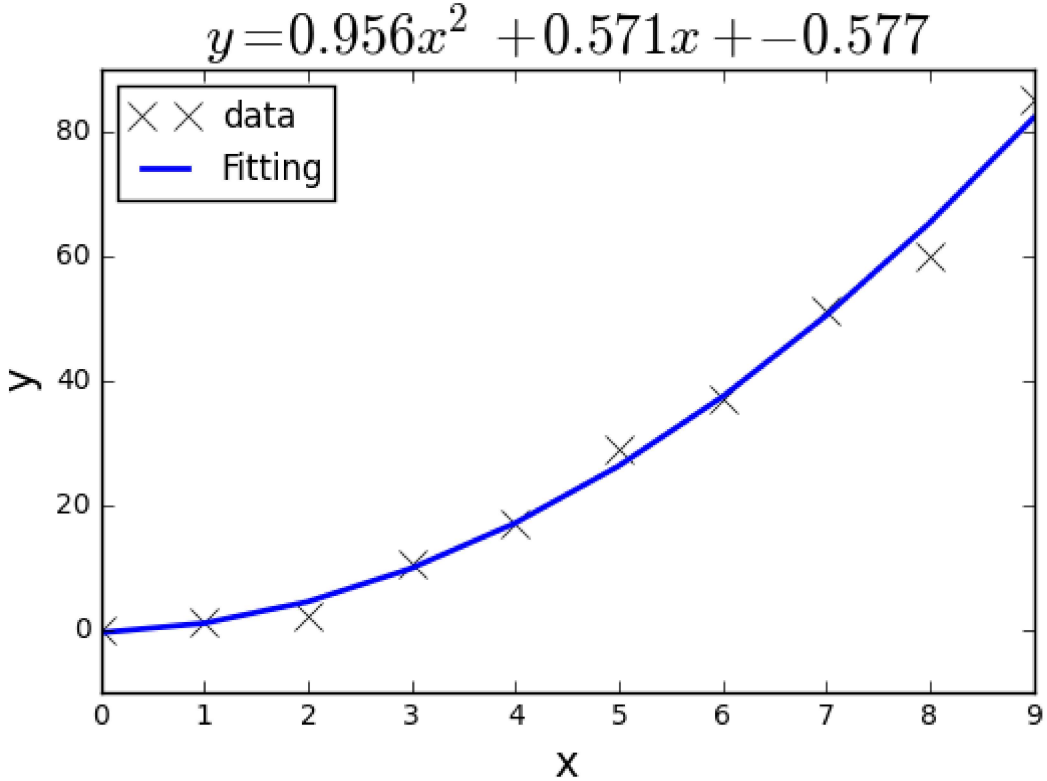
In [55]:
```
p = np.poly1d(z) # fitting هنا نضع المعاملات التي حصلنا عليها من
#  الان اصبحت p  عبارة عن كثيرة حدود من الدرجة الاولى
# p(x) = a.x + b
pl.plot(x,y,'ko',label = 'Data') #  رسم البيانات التي اخذناها من القراءات اعلاه
pl.plot(x,p(x),label = 'Fitting') # رسم الدالة التي قمنا بعمل fitting لها
pl.legend(loc = 0)
pl.text(4,10,'$y = {:.3f}x + {:.3f}$'.format(*z) , fontsize = 20)
pl.ylabel('y' , fontsize = 18)
pl.xlabel('x', fontsize = 18)
pl.show()
```



In [56]:
```
x = range(0,10)
y = [0,1,2,10.5,17,29,37,51,60,85]
z = np.polyfit(x,y,2) # هذا ايضا fitting لكثيرة حدود لكن من الدرجة الثانية
print 'y = {}x^2 + {}x + {}'.format(*z)
```

```
y = 0.956439393939x^2 + 0.570833333333x + -0.577272727273
```

In [57]:
```
p = np.poly1d(z)
pl.plot(x,y,'kx',label = 'data',markersize = 10)
pl.plot(x,p(x),label = 'Fitting',lw = 2)
pl.title('$y = {:.3f}x^2 + {:.3f}x + {:.3f}$'.format(*z),fontsize = 20)
pl.xlabel('x',fontsize = 15)
pl.ylabel('y',fontsize = 15)
pl.legend(loc = 0)
pl.show()
```



9- Making functions Interactive: جعل الدوال تفاعلية

In [58]:
```
from ipywidgets import interact # يعمل في jupyter notebook فقط
```

In [59]:
```
def f(x): # دالة تقوم بإرجاع نفس الرقم المدخل
    return x
interact(f,x = (1,10,0.1))# يبدأ المتغير من 1 وينتهي عند 10 وكل خطوة تساوي 0.1
```

```
4.9
```

Out[59]:
```
<function __main__.f>
```

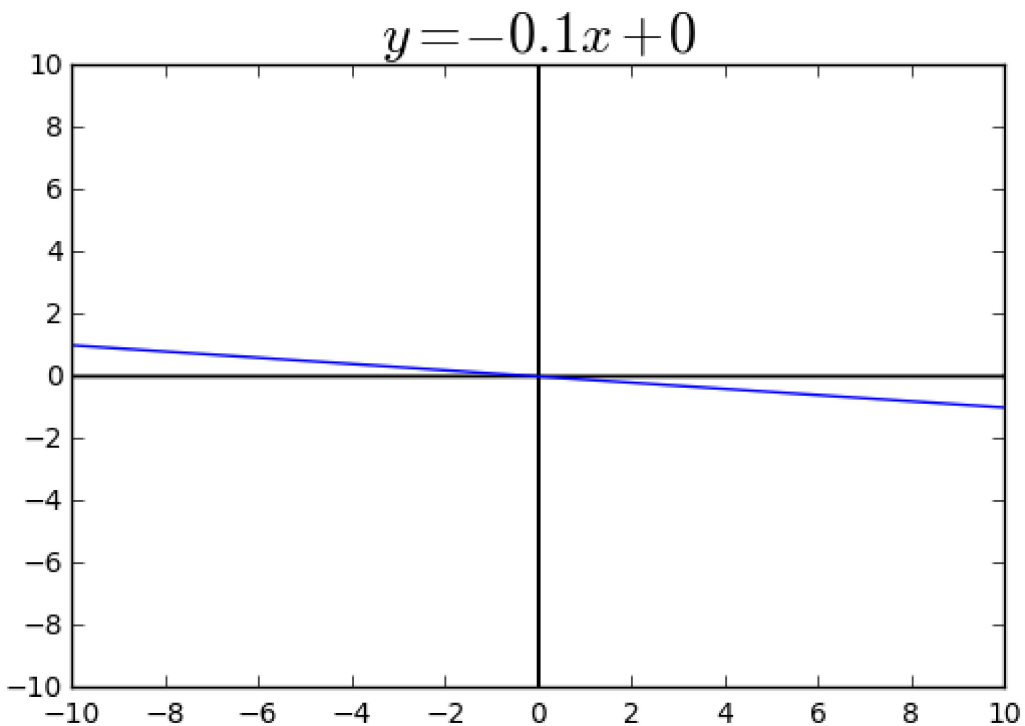
In [60]:
```
KE = lambda m,v: 'KE = {} J'.format(0.5 * m * v**2)
interact(KE, m = (1,10,0.5),v = (1,100))
```

```
'KE = 6250.0 J'
```

Out[60]:
```
<function __main__.<lambda>>
```

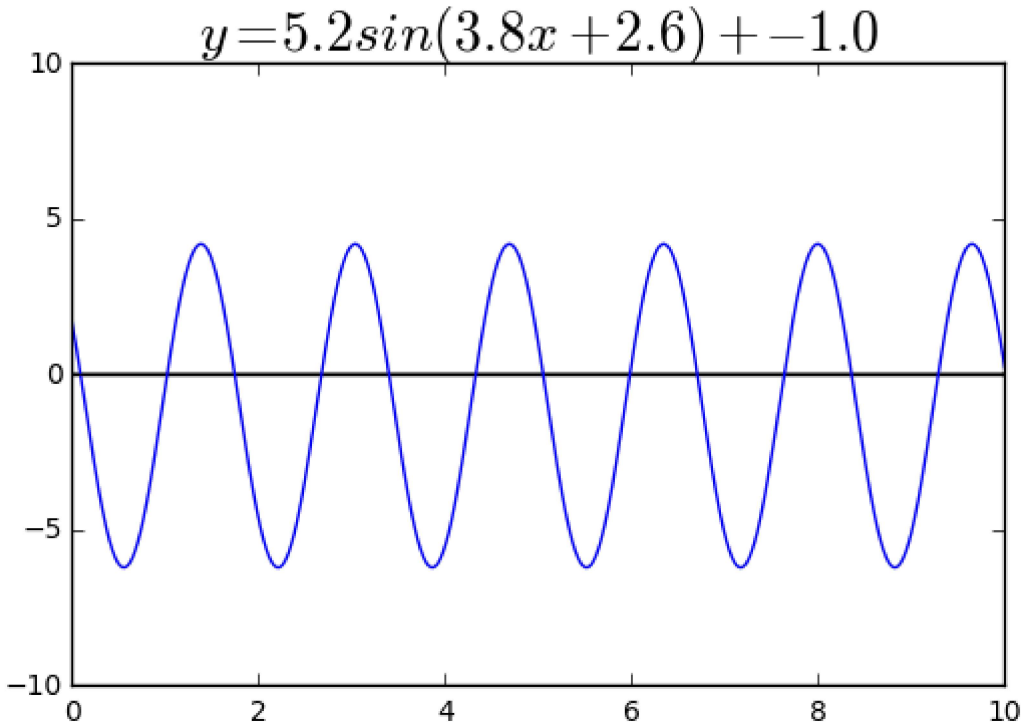
In [61]:
```
def line(a,b):
    x = np.arange(-15,15)
    y = a * x + b
    pl.plot(x,y)
    pl.arrow(-10,0,20,0)
    pl.arrow(0,-10,0,20)
    pl.xlim(-10,10)
    pl.ylim(-10,10)
    pl.xticks(np.linspace(-10,10,11)) # x يقوم بتقسيم محور
    pl.yticks(np.linspace(-10,10,11)) # y يقوم بتقسيم محور
    pl.title('$y = {}x + {}$'.format(a,b),fontsize = 20)

interact(line, a = (-5,5,0.1), b = (-5,5,1))
```

Out[61]: <function __main__.line>



In [62]:
```
def sin_fun(a,b,c,d):
    x = np.linspace(0,10,1000)
    y = a * np.sin(b * x + c) + d
    pl.plot(x,y)
    pl.arrow(0,0,10,0)
    pl.xlim(0,10)
    pl.ylim(-10,10)
    pl.title('$y = {} sin({}x + {} ) + {}$'.format(a,b,c,d),fontsize = 20)
```

In [63]:
```
interact(sin_fun, a = (0,10,0.2), b = (0,10,0.1), c = (-10,10,0.1) , d = (-10,10,0.1) )
```



11- Tabels

In [64]:
```
import pandas as pd
# هذه المكتبة مفيدة في صنع والتعامل مع الجداول
pd.options.display.max_rows = 7
```

In [65]:
```
a = pd.Series([0,1,2,3,4,5,6]) #يجعل التعامل مع الجداول اكثر سهولة array يشبه Series
a # له بعد واحد فقط، أي لا يمكن عمل أكثر من عمود Series
```

Out[65]:
```
0    0
1    1
2    2
3    3
4    4
5    5
6    6
dtype: int64
```

In [66]:
```
2 * a
```

Out[66]:
```
0     0
1     2
2     4
3     6
4     8
5    10
6    12
dtype: int64
```

In [67]:
```
a > 4
```

Out[67]:
```
0    False
1    False
2    False
3    False
4    False
5     True
6     True
dtype: bool
```

In [68]:
```
a.describe() # هذه الدالة تعطيك وصف للبيانات الموجودة في a
```

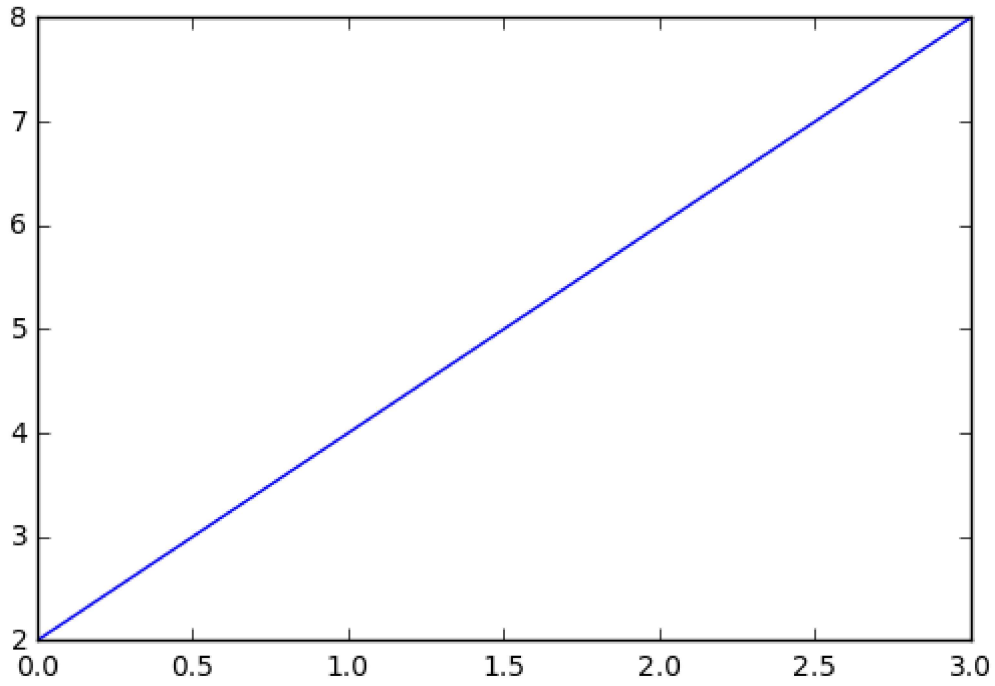
Out[68]:
```
count    7.000000
mean     3.000000
std      2.160247
           ...
50%      3.000000
75%      4.500000
max      6.000000
dtype: float64
```

In [69]:
```
# يمكنك ان تصنع جدولا من خلال دالة DataFrame
data = [2,4,6,8]
df = pd.DataFrame(data, columns = ['a'])
df # اسم الجدول
```

Out[69]:

	a
0	2
1	4
2	6
3	8

In [70]:
```
df['a'].plot() # رسم بياني سريع للعمود a
pl.show()
```



In [71]:
```
df['a'][1]
```

Out[71]: 4

In [72]:
```
print df['a'][2:]
```

```
2    6
3    8
Name: a, dtype: int64
```

In [73]:
```
df['b'] = [10,33,42,100] # b اضافة عمود جديد باسم
df
```

Out[73]:

	a	b
0	2	10
1	4	33
2	6	42
3	8	100

In [74]:
```
df['c = a + b'] =df['a']+df['b']
df
```

Out[74]:

	a	b	c = a + b
0	2	10	12
1	4	33	37
2	6	42	48
3	8	100	108

In [75]:
```
df['b'][0] = 20   # b تغيير قيمة عنصر في العمود
df
```

Out[75]:

	a	b	c = a + b
0	2	20	12
1	4	33	37
2	6	42	48
3	8	100	108

In [76]:
```
df['R']= df['c = a + b']<50
df
```

Out[76]:

	a	b	c = a + b	R
0	2	20	12	True
1	4	33	37	True
2	6	42	48	True
3	8	100	108	False

In [77]:
```
del df['R']  # حذف عمود
```

In [78]:
```
df
```

Out[78]:

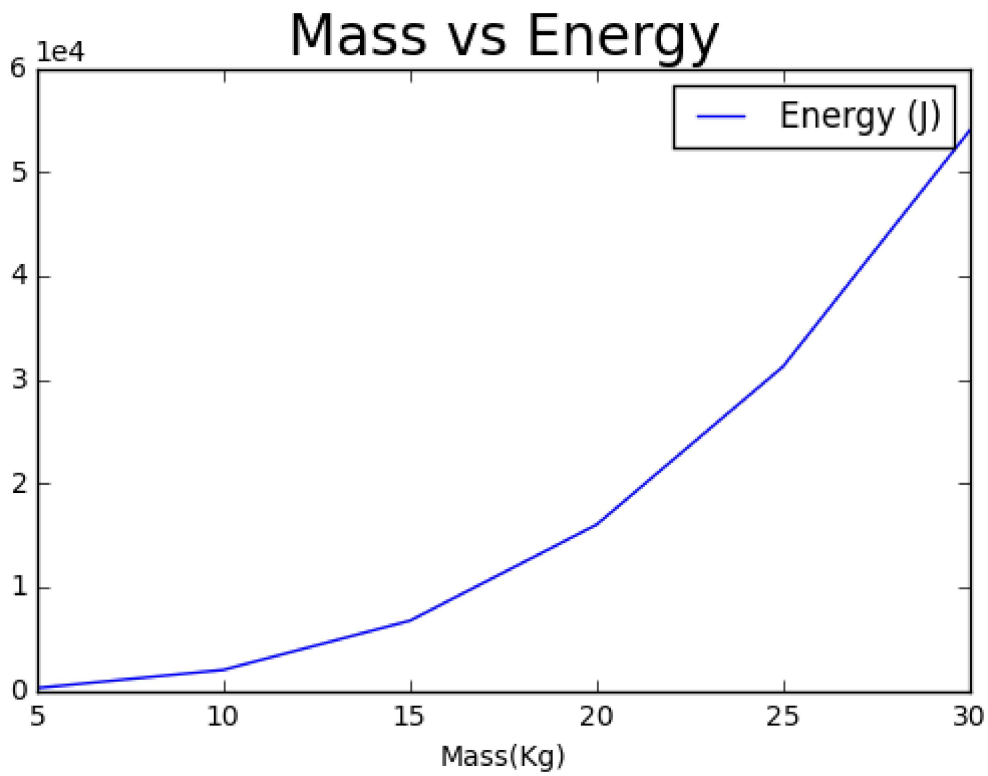
	a	b	c = a + b
0	2	20	12
1	4	33	37
2	6	42	48
3	8	100	108

In [79]:
```
KE = lambda m,v: 0.5 * m * v**2

mass = np.array([5,10,15,20,25,30])
velocity = np.array([10,20,30,40,50,60])
k = KE(mass,velocity)
data = {'Mass(Kg)':mass,'Velocity (m/s)':velocity,'Energy (J)': k} #وضع البيانات على شكل فهرس
df = pd.DataFrame(data)
df[['Mass(Kg)','Velocity (m/s)','Energy (J)']] # يقوم بترتيب الأعمدة
```

Out[79]:

	Mass(Kg)	Velocity (m/s)	Energy (J)
0	5	10	250
1	10	20	2000
2	15	30	6750
3	20	40	16000
4	25	50	31250
5	30	60	54000

In [80]:
```
df.plot('Mass(Kg)','Energy (J)')
pl.title('Mass vs Energy',fontsize = 20)
pl.ticklabel_format(axis='y', scilimits=(0,0)) # يختصر رياضيا الارقام التي على محور y
pl.show()
```



In [81]:
```
df.to_excel('C:/Users/User/Desktop/table.xlsx') # يقوم بحفظ الجدول إلى ملف الإكسل
```

In [82]:
```
pd.read_excel('C:/Users/User/Desktop/table.xlsx') # يقوم بقراءة ملف أكسل
```

Out[82]:

	Energy (J)	Mass(Kg)	Velocity (m/s)
0	250	5	10
1	2000	10	20
2	6750	15	30
3	16000	20	40
4	31250	25	50
5	54000	30	60

12- Time:

In [83]:
```
from time import time, localtime ,sleep
# time, localtime ,sleep استدعي الدوال التالية  time من مكتبة
```

In [84]:
```
time()# يعطيك الوقت الحالي بالثواني، باعتبار ان الزمن بدأ عام 1970 ميلادي
```

Out[84]:
```
1481557537.804
```

In [85]:
```
# من فوائد استخدام هذه الدالة، حساب مدة استغراق حادثة ما
t1 = time()  # عندما يعمل هذا السطر يتم حساب الزمن ووضع في t1

for i in range(100000):  # هذه العملية تستغرق زمن ما حتى تنتهي
    y = i**2

t2 = time() # بعد انتهاء العملية السابقة، يعمل هذا السطر ويسجل الزمن من جديد ويضعه في t2
total_time = t2 - t1   # الزمن المستغرق
total_time # النتيجة بالثواني
```

Out[85]:
```
0.10899996757507324
```

In [86]:
```
localtime() # دالة تعطيك الوقت الحالي بالتفصيل
```

Out[86]:
```
time.struct_time(tm_year=2016, tm_mon=12, tm_mday=12, tm_hour=18, tm_min=45, tm_sec=38, tm_wday=0, tm_yday=34
7, tm_isdst=0)
```

In [87]:
```
# تستفيد منها في كتابة التاريخ والوقت بدقة، وقت إجراء التجربة مثلاً
date_time = 'Date: {}/{}/{} \nTime: {}:{}:{}'.format(localtime()[0],localtime()[1],\
                                                     localtime()[2],localtime()[3],\
                                                     localtime()[4],localtime()[5])
print date_time
```

```
Date: 2016/12/12
Time: 18:45:38
```

In [88]:
```
for i in range(4):
    print i
    sleep(1) # تقوم بإيقاف الحاسب لمدة محددة باثواني
```

```
0
1
2
3
```

13 - Numurical integration and derivative:

التفاضل والتكامل العددي يستخدم عندما نتعامل مع دوال لا يمكن لنا تكاملها أو تفاضلها بالطرق التقليدية
ويكون الحل عبارة عن أرقام وليس دالة
ويمكن تقسيمه إلى قسمين:
A- تكامل وتفاضل للدوال
B- تكامل وتفاضل للبيانات

A) For Equations

التكامل والتفاضل العددي للدوال

In [89]:
```
f = lambda x: x**2   # f(x) = x^2

def derivative(x):  # x يعطيك المشتقة (الميل) عند النقطة
    dx = 0.0001    #  dx = x2 - x1
    dy = f(x+dx)-f(x) # dy = y2 - y1
    return dy/dx

#  2x هي  x^2 للتأكد، نحن نعلم ان مشتقة
print derivative(2) # 4=2*2 أي مشتقة الدالة f(x) عند النقطة2: الاجابة المتوقعة ستكون
print derivative(3)
print derivative(4)
```

```
4.00010000001
6.00010000001
8.00009999999
```

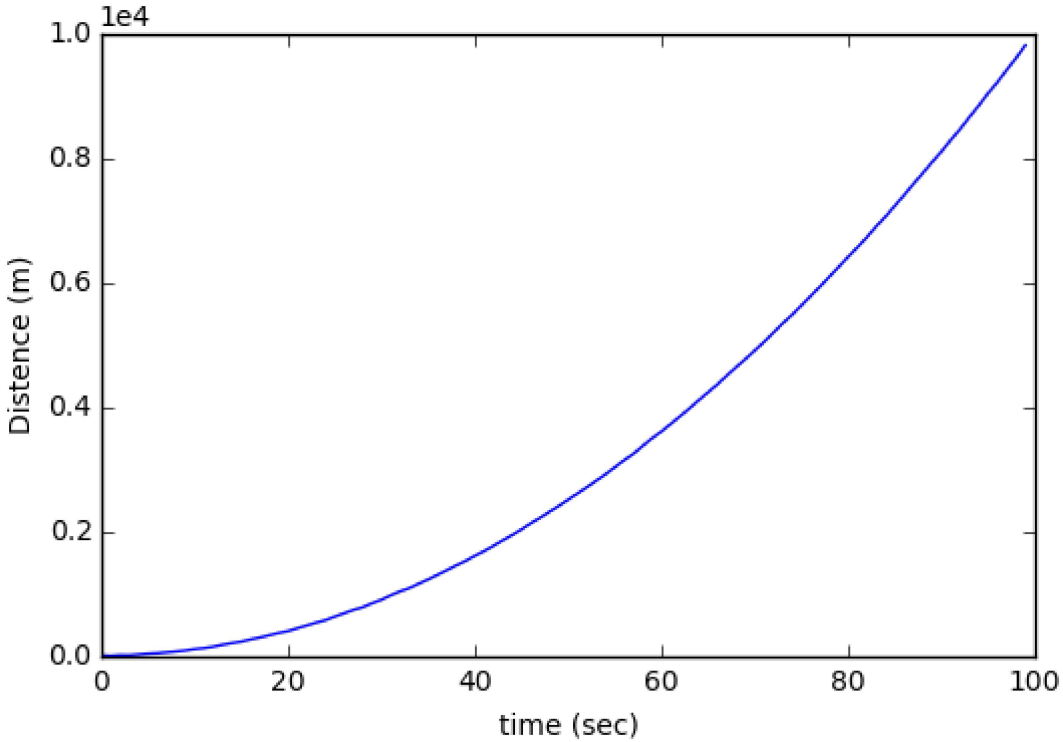
In [90]:
```
#  لحساب التكامل لدالة ما، أو حساب المساحة تحت المنحنى. تقسم المساحة الى مستطيلات
# صغيرة جدا، يمكن حساب مساحة كل واحدة منها من خلال ضرب عرضها في طولها
#وبعد ذلك يتم جمع مساحاتها كلها وهو قيمة التكامل

def integral(a,b,n): # n يعطيك التكالمل للدالة f(x) من النقطة a إلى النقطة b بعدد مستطيلات
    width = float(b-a)/n # عرض كل مستطيل صغير
    tot_area = 0  # مجموع مساحات المستطيلات قبل اجراء الحسابات
    for i in range(n): # يتم من خلال هذه الحلقة حساب مساحة كل مستطيل على حده
        hight = f(a + i*width)  # حساب الارتفاع للمستطيل ذي الرقم i
        area = width*hight      # حساب مساحة المستطيل i (الطول في العرض)
        tot_area += area      # هنا نضيف مساحة المستطيل i إلى مجموع المساحات
    return tot_area        # المساحة الكلية او قيمة التكامل

integral(0,1,1000)  #  x = 1 إلى  x = 0 من  x^2 كامل الدالة
# من خلال تقسيم المساحة تحت المنحنى الى 1000 مستطيل
# ملاحظة: كل ما زاد عدد المستطيلات زادت الدقة
# (x^3)/3 يساوي  x^2 للتأكد: تكامل الدالة
# 1/3 = 0.333333  بالتالي التكامل المحدود من صفر إلى واحد هو
```

Out[90]:
```
0.33283350000000034
```

For Data:

التكامل والتفاضل للبيانات

In [91]:
```
# لنفترض ان لدينا بيانات لسيارة تتحرك في بعد واحد
data = np.loadtxt('C:/Users/User/Desktop/Computation Physics/time_distance.txt')
t = data[:,0]
d = data[:,1]
pl.plot(t,d)
pl.ylim(0)
pl.xlabel('time (sec)')
pl.ylabel('Distence (m)')
pl.ticklabel_format(axis='y', scilimits=(0,0))
```



In [92]:

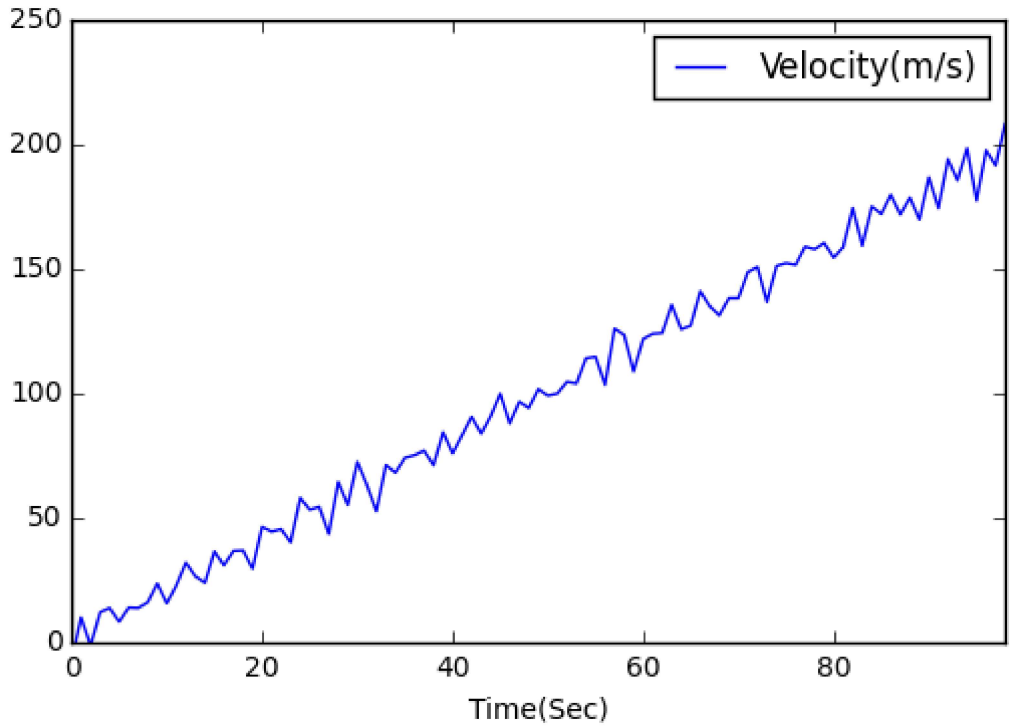
```
# لكي نوجد السرعة اللحضية للسيارة، علينا أن نقوم بالاشتقاق للبيانات عند لحضة معينة
def data_derivative(x,y): # تعريف دالة تقوم باشتقاق محور x و y
    slope = [] # الميل لكل نقطة بيانات قبل اجراء الاشتقاق
    for i in range(len(x)-1): # هذه الحلقة تقوم بالدوران على كل البيانات
        s = (y[i+1]-y[i])/(x[i+1]-x[i]) # الميل عند البيانات i هو عبارة عن (y2-y1/x2-x1)
        slope.append(s)  # هنا نضيف الميل عند البيانات i إلى الميل الكلي
    return slope # التفاضل عند جميع نقاط البيانات

v = data_derivative(t,d) # السرعة تساوي مشتقة الازاحة بالنسبة للزمن
# ملاحضة: أخر نقطة في البيانات لا نستطيع ايجاد المشتقة عندها، لاننا لا نعلم قيمة النقطة التي تليها
df = pd.DataFrame({'Time(Sec)':t[:-1] , 'Distance(m)': d[:-1] , 'Velocity(m/s)': v })
df[['Time(Sec)','Distance(m)','Velocity(m/s)']] # هذا السطر يرتب الأعمدة حسب ما تريد
```

Out[92]:

	Time(Sec)	Distance(m)	Velocity(m/s)
0	0	4.377896	-7.886654
1	1	-3.508758	9.725491
2	2	6.216733	-1.611223
...	...	...	...
96	96	9209.227330	197.305946
97	97	9406.533277	191.174729
98	98	9597.708005	207.939082

99 rows × 3 columns

In [93]:
```
df.plot('Time(Sec)','Velocity(m/s)',ylim = 0 )

#pl.fill_between(t[:-1],v ) # شغل هذا السطر وانظر ماذا يحصل؟

pl.show()
```



In [94]:
```
# التكامل العديدي للبيانات يعطينا المساحة الموجود تحت منحنا البيانات
def data_integration(x,y): # تعريف دالة تقوم بتكامل البيانات ذات محور x و y
        R = []  # المساحة الكلية
        for i in range(len(x)-1): # يدور على جميع البيانات
            r = (x[i+1]-x[i])*y[i] # مساحة المستطيل = y*(x2-x1) الذي يقابل x1
            R.append(r) # اضف مساحة المستطيل الصغير الى المساحة الكلية
        tot_area = sum(R) # إجمع جميع المستطيلات الصغيرة
        return tot_area # المساحة الكلية او قيمة التكامل
tot_distence = data_integration(t,v) # التكامل العددي للسرعة بالنسبة للزمن
tot_distence  # المسافة الكلية التي قطعتها السيارة
```

Out[94]: 9801.2691908973793

14- Else you can do with python

In [95]:
```
# هذه المكتبة تسهل عليك اخذ اسم مسار الملفات من الحاسب
from Tkinter import Tk
from tkFileDialog import askdirectory,askopenfilename,askopenfilenames
```

In [96]:
```
Tk().withdraw()
path = askdirectory()  # دالة تعطي مسار المجلد
```

In [97]:
```
path = askopenfilename() # دالة تعطي مسار ملف واحد فقط
path
```

Out[97]:
```
u'C:/Users/User/Desktop/Computation Physics/Computational Physics Exercises part 1.pdf'
```

In [98]:
```
path = askopenfilenames() # دالة تعطي مسار أكثر من ملف
path
```

Out[98]:
```
(u'C:/Users/User/Desktop/Computation Physics/data.txt',
 u'C:/Users/User/Desktop/Computation Physics/note.txt',
 u'C:/Users/User/Desktop/Computation Physics/computational physics by python part 1.pdf',
 u'C:/Users/User/Desktop/Computation Physics/computational physics by python SymPy.ipynb',
 u'C:/Users/User/Desktop/Computation Physics/Computational Physics Exercises part 1.pdf')
```

In [99]:
```
#اختيار اسم الملف من المسار
file_name = 'C:/Users/User/Desktop/Computation Physics/Computational Physics Exercises part 1.pdf'.split('/')[-1
file_name
```

Out[99]:
```
'Computational Physics Exercises part 1.pdf'
```

In [100]:
```
# اسم الملف من دون الصيغية
file_name[:file_name.index('.')]
```

Out[100]:
```
'Computational Physics Exercises part 1'
```