Graphiques avec Mathplotlib

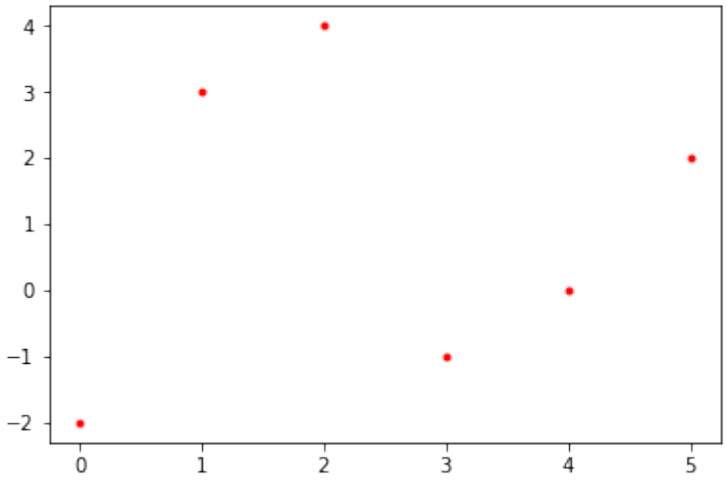
Documents et compléments disponibles sur www.math93.com.

On importe le sous-module pyplot de matplotlib qu'on renomme au passage plt, par commodité.

1 Afficher un nuage de points

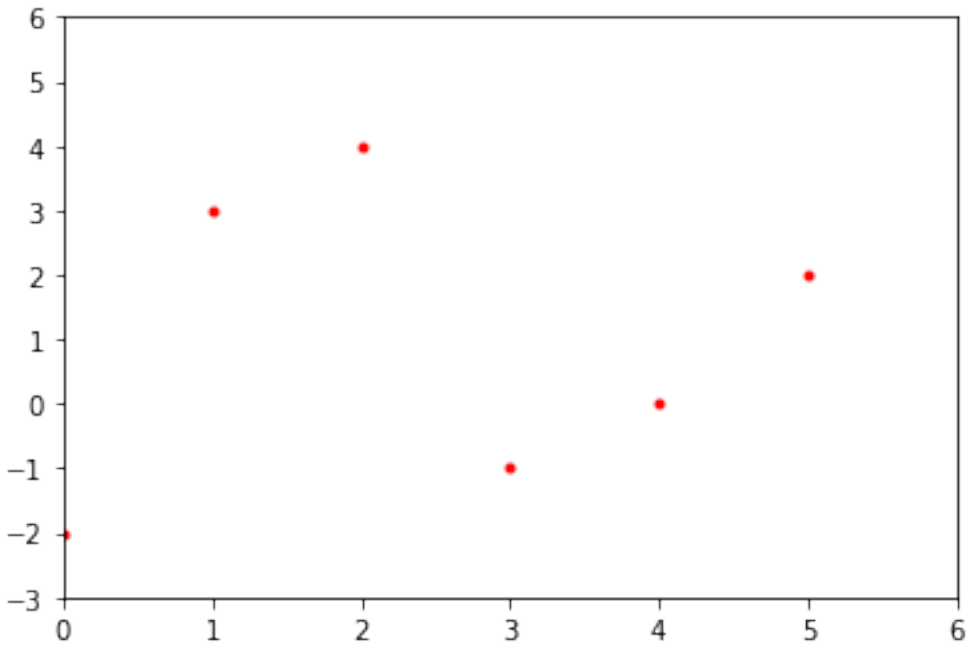
On peut facilement tracer les nuages de points correspondants à une fonction définie, en utilisant les fonctions du module matplotlib. Sous repl ou de nombreux logiciels online, il faut convertir le fichier en image avec l'instruction fig = plt.figure() et la sauvegarder sous un nom avec l'instruction fig.savefig('graph.png').

```
In [7]: import matplotlib.pyplot as plt
        fig = plt.figure() # nécessaire sur repl.it
        #
        vx=[0,1,2,3,4,5]
        vy=[-2,3,4,-1,0,2]

        plt.plot(vx,vy,'.',color='red') # le 3e argument '.' pour un nuage de points
        #plt.plot(vx,vy,'-',color='red') #   ou '-' pour les relier

        fig.savefig('graph.png') # nécessaire sur repl.it, sinon simplement plt.show()
```



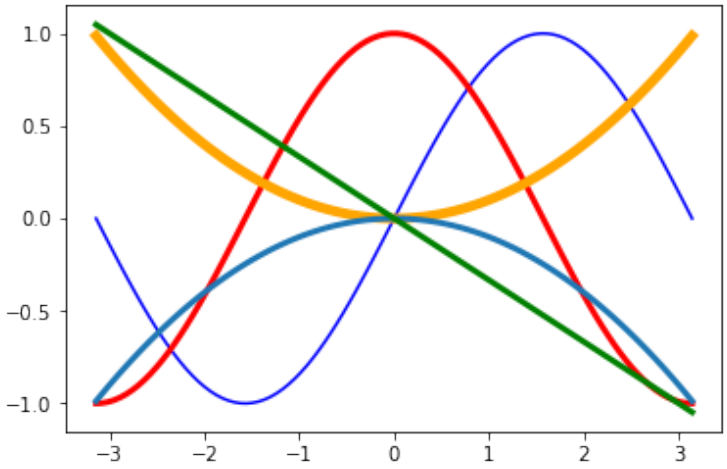
On peut aussi préciser les axes :

```
In [6]: import matplotlib.pyplot as plt
        fig = plt.figure() # nécessaire sur repl.it

        vx=[0,1,2,3,4,5]
        vy=[-2,3,4,-1,0,2]

        axes = plt.gca() # pour travailler sur les axes
        axes.set_xlim(0, 6)
        axes.set_ylim(-3, 6)

        plt.plot(vx,vy,'.',color='red')
        #plt.plot(vx,vy,'-',color='red') #   ou '-' pour les relier

        fig.savefig('graph.png') # nécessaire sur repl.it, sinon simplement plt.show()
```



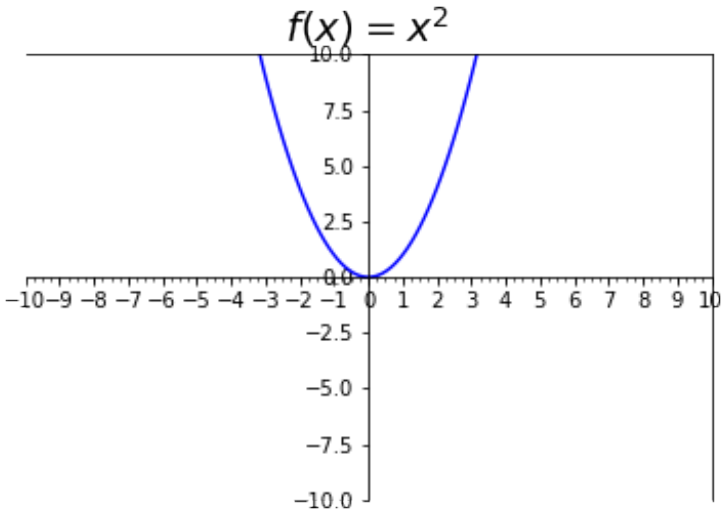
2 Courbes représentatives de fonctions

In [28]:
```
##Courbes représentatives de fonctions
# Dans l'exemple suivant, on construit une courbe représentative
# avec une fonction définie de plusieurs façons .

import matplotlib.pyplot as plt
import numpy as np
from math import *

def f(x):
    return x*x/10
def g(x):
    return -x**2/10
def h(x):
    return -x/3

X = np.linspace(-np.pi, np.pi, 100) # une liste de 100 valeurs entre -pi et pi

Y = np.sin(X) # ne fonctionne pas avec sin(X), il faut utiliser np.sin(X)
Z = [f(i) for i in X]
W = g(X) # ne fonctionne pas avec cos(X), il faut utiliser np.cos(X)
#
plt.plot(X, Y, color="blue", linewidth=1.5, linestyle="-")
plt.plot(X, np.cos(X),color="red", linewidth=3)
plt.plot(X, [f(x) for x in X],color="orange", linewidth=5)
plt.plot(X, W, linewidth=3)
# si on ne définit pas de couleur, elle est automatique
plt.plot(X, h(X),color="green", linewidth=3)

plt.savefig("graph") # sauvegarde l'image sous le nom graph.png
plt.show()           # affiche l'image à l'écran
```



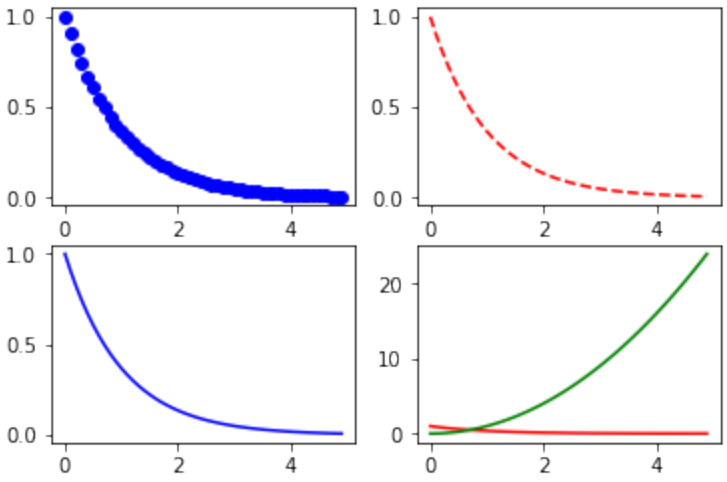
On peut aussi redéfinir les axes :

```
In [36]: import numpy as np
         import matplotlib.pyplot as plt
         from matplotlib.ticker import MultipleLocator, FormatStrFormatter
         fig = plt.figure()# pour repl.it

         x = np.linspace(-6., 6., 100) # permet de créer une liste de 100 réels de -6 à 6
         y = x ** 2

         plt.plot(x, y ,'-', c = 'b')
         plt.title(r'$f(x) = x^2$',fontsize = 20)

         #plt.xlabel('x',fontsize = 16)
         #plt.ylabel('$x^2$', fontsize = 16)

         plt.xlim([-10, 10])
         plt.ylim([-10, 10])

         ax = plt.gca()
         ax.xaxis.set_major_locator(MultipleLocator(1))
         ax.xaxis.set_minor_locator(MultipleLocator(0.25))

         #plt.axhline(color = 'k') # axe des abscisse
         #plt.axvline(color='k') # axe des ordonnées

         ax.spines['bottom'].set_position(('data',0)) # pour redéfinir l'axe des abscisses
         ax.spines['left'].set_position(('data',0)) # pour redéfinir l'axe des ordonnées

         fig.savefig('graph.png') # ou  plt.show() sur Spyder
```



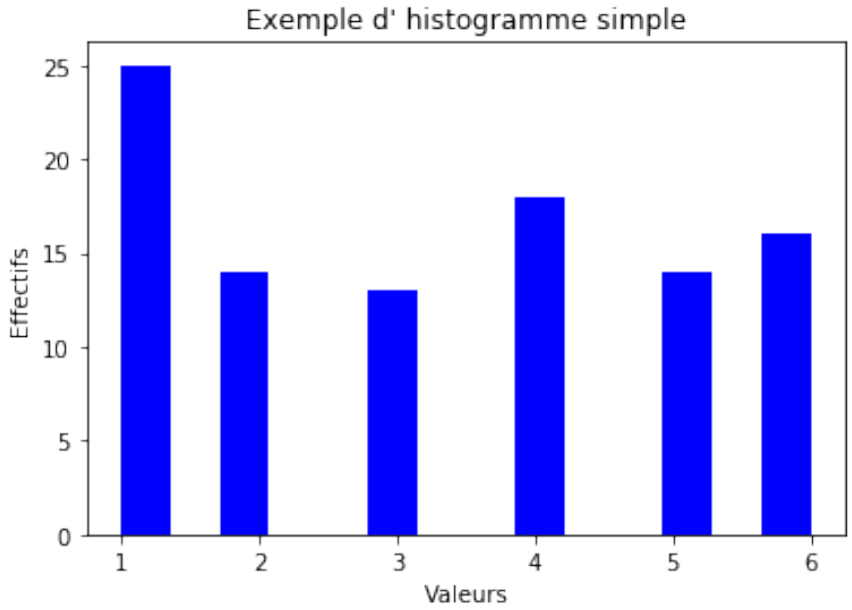
On peut aussi vouloir obtenir plusieurs courbes.
Dans la commande plt.subplot l'argument est nbre de lignes, nbre de colonnes, numéro de la figure. Il y a une condition à respecter : le nombre de lignes multiplié par le nombre de colonnes est supérieur ou égal au nombre de figure. Ensuite Matplotlib place les figures au fur et à mesure dans le sens des lignes.

In [54]:
```
# mathplotlib : plusieurs courbes

import numpy as np
import matplotlib.pyplot as plt

def f(t):
   return np.exp(-t)

t1 = np.arange(0.0, 5.0, 0.1)
#  Un peu comme range mais avec un pas qui peut être décimal.

plt.subplot(221) # 1re figure
plt.plot(t1, f(t1), 'bo')

plt.subplot(222) # 2e figure
plt.plot(t1, f(t1), 'r--')

plt.subplot(223) # 3e figure
plt.plot(t1, f(t1), 'b-')

plt.subplot(224) # 4e figure  avec 2 courbes
plt.plot(t1, f(t1),  'k',color='red')
plt.plot(t1, t1**2,  'k',color='green')
plt.savefig("graph")
```



3 Histogramme

In [50]:
```
# Histogramme

import numpy as np
import random
import matplotlib.pyplot as plt
fig = plt.figure()# pour repl.it

#
def echantillon(n):
    return [random.randint(1,6) for i in range(n)]
#

fig = plt.figure()
x=echantillon(100)

plt.hist(x,14, density=0, facecolor='b',align='mid')
# Le 2e argument donne l'espace entre les données
# density = 1 : pour les fréquences
# density = 0 : pour les occurences
fig.savefig('graph.png') # ou  plt.show() sur anaconda

plt.xlabel('Valeurs')
plt.ylabel('Effectifs')
plt.title('Exemple d\' histogramme simple')

fig.savefig('graph.png') # ou  plt.show() sur Spyder
```



4 Diagramme en batons

Il est aussi possible de tarcer des diagrammes en bâtons comme on le voit dans le programme suivant.
Ici, on lance deux dés à 6 faces et on enregsistre dans la liste L, l'effectif pour chaque valeur obtenue de 0 à 12 (évidemment 0 et 1 restent à 0 mais le programme est plus simple à écrire ainsi).

```
In [32]: from matplotlib.pyplot import *
         from random import *

         X = [0,1,2,3,4,5,6,7,8,9,10,11,12]
         L = [0,0,0,0,0,0,0,0,0,0,0 ,0 ,0 ]
         for i in range(100):
           double_de=randint(1,6)+randint(1,6)
           L[double_de]=L[double_de]+1

         bar(X,L)
         savefig("diagramme")
         show()
```