Modulo matplotlib

El modulo matplotlib es un paquete que forma parte de la librería SciPy para calculo en ciencia e ingeniería y permite hacer gráficos. Esta librería tiene muchos métodos y algunos son muy elaborados y específicos como para ser abordados en su totalidad. Por lo que nos restringiremos a revisar las funcionalidades básicas.

Lo primero que debemos hacer es importar el modulo. Lo que requerimos es importar el modulo del paquete matplotlib:

In [1]:
```
from matplotlib import pyplot as plt
```

Esto se suele abreviar de la forma

In [3]:
```
import matplotlib.pyplot as plt
```

Otro detalle importante es el uso de "funciones magicas". Estas son funciones especiales de Jupyter Notebook para el control de las propiedades del notebook. En concreto, para mantener el gráfico generado en el mismo notebook y evitar la instrucción plt.show() para mostrar el gráfico, se utiliza:

In [6]:
```
%matplotlib inline
```

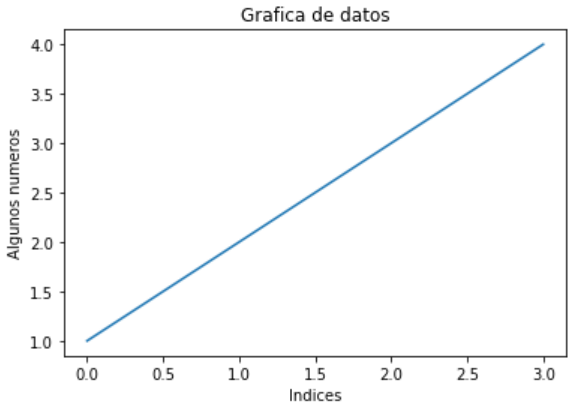
La forma usual de graficar los valores de un arreglo es:

```
plt.plot(x, y)     # Esto genera el objeto grafico (arreglo x vs. arreglo y)
plt.show()         # Esto muestra la grafica
```

Sin embargo, utilizando %matplotlib inline ya no es necesario incluír la linea plt.show() (Aunque si no se utiliza aparcera el resultado de la instrucción):

In [9]:
```
plt.plot([1, 2, 3, 4])
plt.title("Grafica de datos")
plt.ylabel("Algunos numeros")
plt.xlabel("Indices")
```

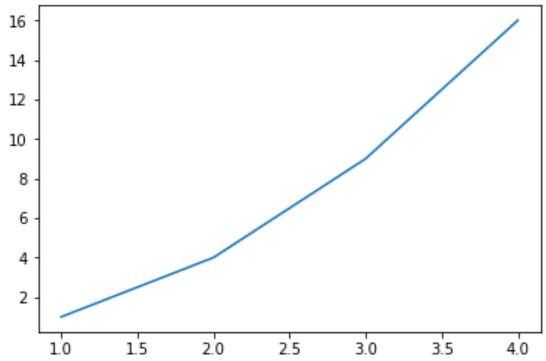
```
<matplotlib.text.Text at 0x1d28cb13c88>
```



Como se puede observar, en el grafico anterior se han graficado los valores de x contra sus indices. Se puede graficas un arreglo x contra u arreglo y:

In [10]:

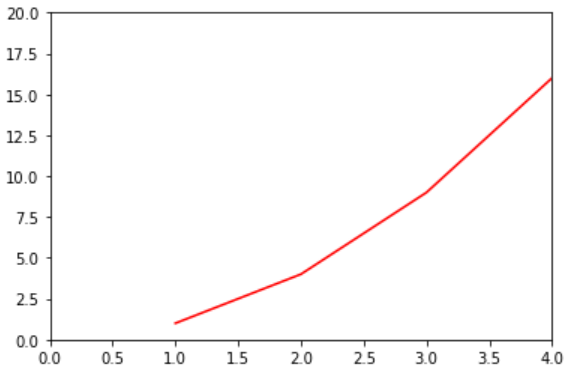
```
plt.plot([1, 2, 3, 4],[1, 4, 9, 16])
plt.show()
```

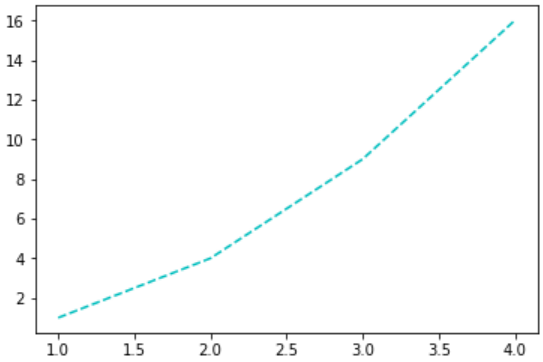


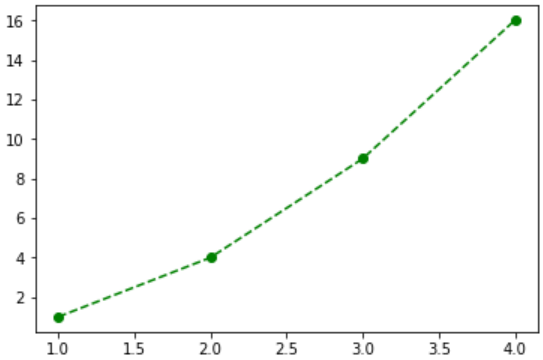
Existe un tercer argumento opcional que permite especificar con un str el color de a linea y el marcador:

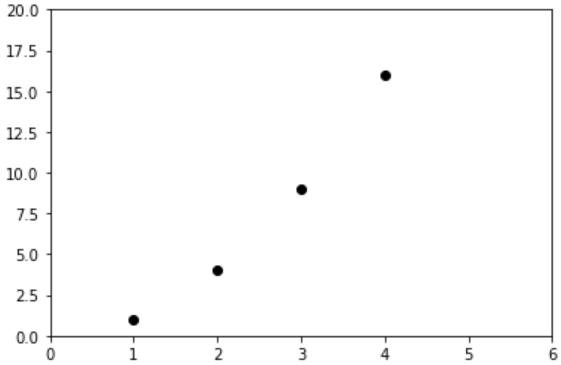
In [15]:

```
# Grafico rojo
plt.plot([1, 2, 3, 4],[1, 4, 9, 16], 'r')
plt.axis([0, 4, 0, 20])
plt.show()

# Grafico cyan con linea discuntinua
plt.plot([1, 2, 3, 4],[1, 4, 9, 16], '--c')
plt.show()

# Grafico verde con linea discontinua y marcadores redondos
plt.plot([1, 2, 3, 4],[1, 4, 9, 16], '--go')
plt.show()

# Grafico negro de solo marcadores
# Ademas vamos a ajustar los ejes x: 0 - 6, y: 0 - 20
plt.plot([1, 2, 3, 4],[1, 4, 9, 16], 'ko')
plt.axis([0, 6, 0, 20])
plt.show()
```


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0


16
14
12
10
8
6
4
2
1.0 1.5 2.0 2.5 3.0 3.5 4.0


16
14
12
10
8
6
4
2
1.0 1.5 2.0 2.5 3.0 3.5 4.0


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0 1 2 3 4 5 6

In [21]:

```
# Cadena de formato grafico (estilos de linea y marcadores)
'''
============================================
character       description
============================================
'-'             solid line style
'--'            dashed line style
'-.'            dash-dot line style
':'             dotted line style
'.'             point marker
','             pixel marker
'o'             circle marker
'v'             triangle_down marker
'^'             triangle_up marker
'<'             triangle_left marker
'>'             triangle_right marker
'1'             tri_down marker
'2'             tri_up marker
'3'             tri_left marker
'4'             tri_right marker
's'             square marker
'p'             pentagon marker
'*'             star marker
'h'             hexagon1 marker
'H'             hexagon2 marker
'+'             plus marker
'x'             x marker
'D'             diamond marker
'd'             thin_diamond marker
'|'             vline marker
'_'             hline marker
============================================
'''
```

```
"\n============================================\ncharacter       description\n============================================\n'-'             s
olid line style\n'--'            dashed line style\n'-.'            dash-dot line style\n':'             dotted line style\n'.'             poi
nt marker\n','             pixel marker\n'o'             circle marker\n'v'             triangle_down marker\n'^'             triangle_up marke
r\n'<'             triangle_left marker\n'>'             triangle_right marker\n'1'             tri_down marker\n'2'             tri_up marker
\n'3'             tri_left marker\n'4'             tri_right marker\n's'             square marker\n'p'             pentagon marker\n'*'
star marker\n'h'             hexagon1 marker\n'H'             hexagon2 marker\n'+'             plus marker\n'x'             x marker\n'D'
diamond marker\n'd'             thin_diamond marker\n'|'             vline marker\n'_'             hline marker\n============================
================\n"
```

In [22]:

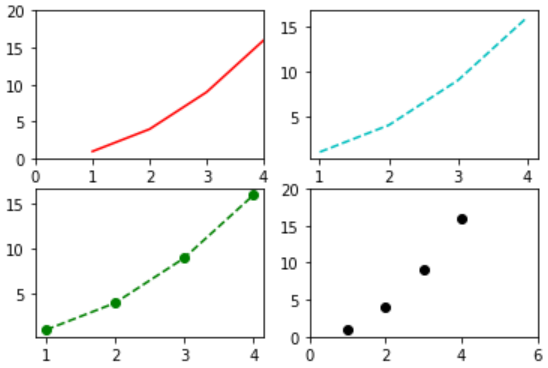
```
#colores :
'''
==================
character   color
==================
'b'         blue
'g'         green
'r'         red
'c'         cyan
'm'         magenta
'y'         yellow
'k'         black
'w'         white
==================
'''
```

```
"\n==================\ncharacter   color\n==================\n'b'         blue\n'g'         green\n'r'         red\n'c'         cyan\n'm'
magenta\n'y'         yellow\n'k'         black\n'w'         white\n==================\n"
```

Estos cuatro graficos se pueden organizar en una sola "figura" qe ha sido dividida como si fuera un arreglo matricial en Filas y Columnas utilizando la instruccion *plt.subplot(m,n,p)*, donde m y n serán las dimensiones de la división y p la posición de la sub-figura:

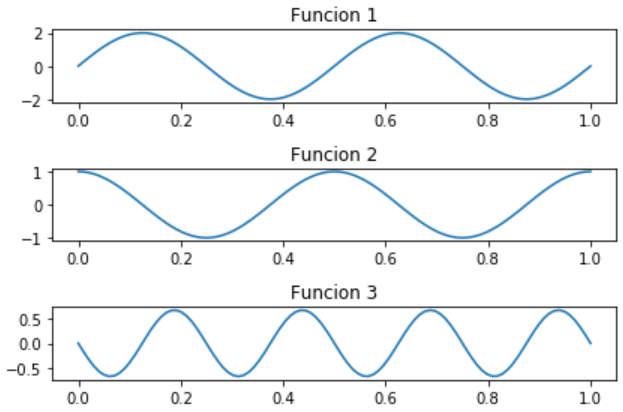
In [23]:

```
plt.figure(1)
# Grafico rojo
plt.subplot(2,2,1)
plt.plot([1, 2, 3, 4],[1, 4, 9, 16], 'r')
plt.axis([0, 4, 0, 20])

# Grafico cyan con linea discuntinua
plt.subplot(2,2,2)
plt.plot([1, 2, 3, 4],[1, 4, 9, 16], '--c')

# Grafico verde con linea discontinua y marcadores redondos
plt.subplot(2,2,3)
plt.plot([1, 2, 3, 4],[1, 4, 9, 16], '--go')

# Grafico negro de solo marcadores
# Ademas vamos a ajustar los ejes x: 0 - 6, y: 0 - 20
plt.subplot(2,2,4)
plt.plot([1, 2, 3, 4],[1, 4, 9, 16], 'ko')
plt.axis([0, 6, 0, 20])
plt.show()
```



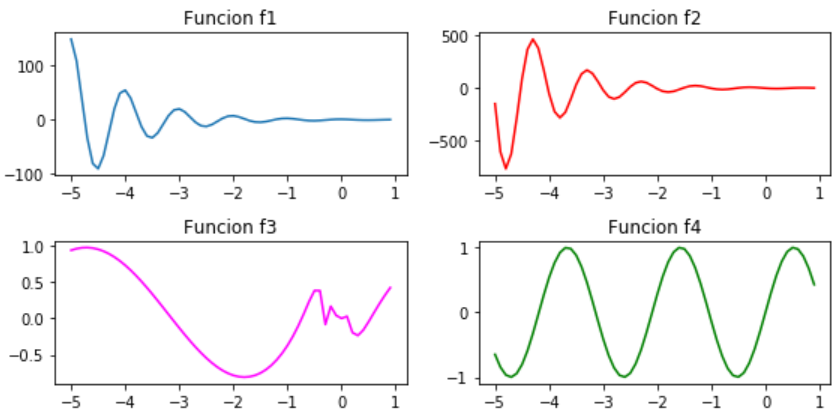
In [24]:

```
# Sub-graficos
A, f = 2, 2   #Amplitud =2, Frecuencia = 5Hz
t = np.linspace(0, 1, 100)
y1 = A * np.sin(2 * np.pi * f * t)
y2 = A/2 * np.cos(2 * np.pi * f * t)
y3 = A/3 * -np.sin(2 * np.pi * 2*f * t)

# Subgrafica superior
plt.subplot(3,1,1)
plt.plot(t, y1)
plt.title("Funcion 1")
# Subgrafica media
plt.subplot(3,1,2)
plt.plot(t, y2)
plt.title("Funcion 2")
#Subgrafica inferior
plt.subplot(3,1,3)
plt.plot(t, y3)
plt.title("Funcion 3")
#Ajuste de las graficas en la ventana
plt.tight_layout()
plt.show()
```



In [25]:

```
# Sub-graficos (instanciando el objeto fig)
def f1(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

def f2(t):
    return -2*np.pi * np.exp(-t) * np.sin(2*np.pi*t) - np.e**(-t)*np.cos(2*np.pi*t)

def f3(t):
    return np.sin(t) * np.cos(1/(t+0.1))

def f4(t):
    return np.sin(3*t)

# Se genera un objeto figura que sera dividido
fig = plt.figure(figsize=(8, 4))
t = np.arange(-5, 1, 0.1)

sub1 = fig.add_subplot(221)       # en lugar de plt.subplot(2,2,1)
sub1.set_title("Funcion f1")
sub1.plot(t, f1(t))

sub2 = fig.add_subplot(222)       # en lugar de plt.subplot(2,2,1)
sub2.set_title("Funcion f2")
sub2.plot(t, f2(t), color='red')

sub3 = fig.add_subplot(223)       # en lugar de plt.subplot(2,2,1)
sub3.set_title("Funcion f3")
sub3.plot(t, f3(t), color='magenta')

sub4 = fig.add_subplot(224)       # en lugar de plt.subplot(2,2,1)
sub4.set_title("Funcion f4")
sub4.plot(t, f4(t), color='green')

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



El método pyplot.plot

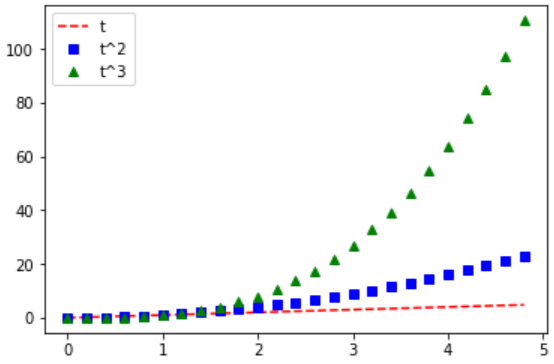
La instruccion plot tiene muchos atributos en cuestion de tipo de linea, color y marcadores ver (http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot (http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot)), sin embargo nos limitremos a utilizar los parametros mas comunes mientras se van mostrando ejemplos.

Se pueden colocar varias graficas en un mismo eje grafico de la forma plt.plot(x1, y1, x2, y2,...):

Vamos a combinar numpy con pyplot para superar ls limitaciones de las listas en cuestion de calculo matematico (de hecho, cuando se pasa una lista a pyplot esta se convierte en un array).

In [26]:

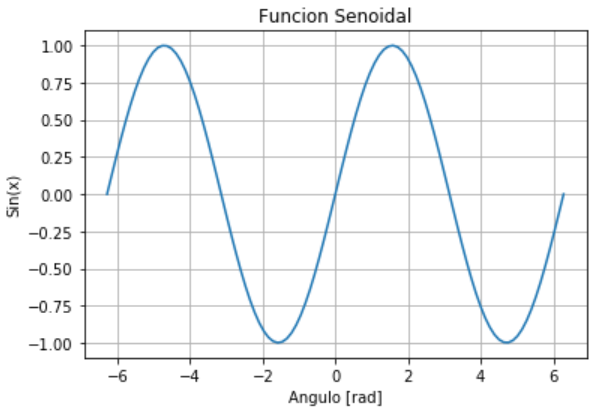
```
import numpy as np
import matplotlib.pyplot as plt

# Un vector de tiempos con intervalos de 200 ms
t = np.arange(0, 5, 0.2)

# Grafico puntos rojo, cuadrados azules  y triangulos verdes
plt.plot(t, t, '--r', t, t**2, 'sb', t, t**3, '^g')
plt.legend(["t", "t^2", "t^3"])
plt.show()
```

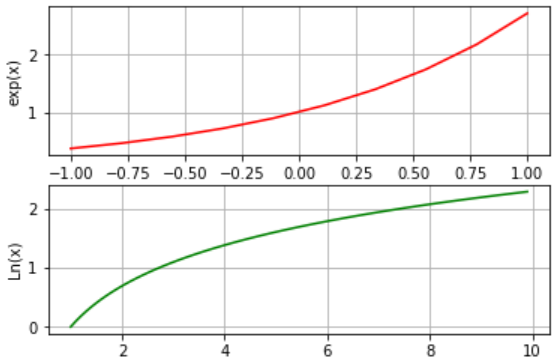


Se pueden combinar multiples figuras y multiples subplots para tener presentaciones graficas multiples, incluyendo titulos, ejes, grilla...

In [27]:

```
# Funcion senoidal
ang = np.linspace(-2*np.pi, 2*np.pi, 100)
y = np.sin(ang)

# Grafica en una sola figura
plt.figure(1)
plt.plot(ang, y)
plt.title("Funcion Senoidal")
plt.ylabel("Sin(x)")
plt.xlabel("Angulo [rad]")
plt.grid()

# Funcion exponencial
x = np.linspace(-1, 1, 10)
y = np.exp(x)

# Grafica en la primera sub-ventana
plt.figure(2)
plt.title("Funciones")
plt.subplot(2, 1, 1)
plt.plot(x, y, 'r')
plt.ylabel("exp(x)")
plt.grid()

# Funcion logaritmo
x = np.arange(1, 10, 0.1)
y = np.log(x + 10e-20)       # Para evitar la division entre 0!!!

# Grafica en la segunda sub-ventana
plt.subplot(2, 1, 2)
plt.plot(x, y, 'g')
plt.ylabel("Ln(x)")
plt.grid()
```

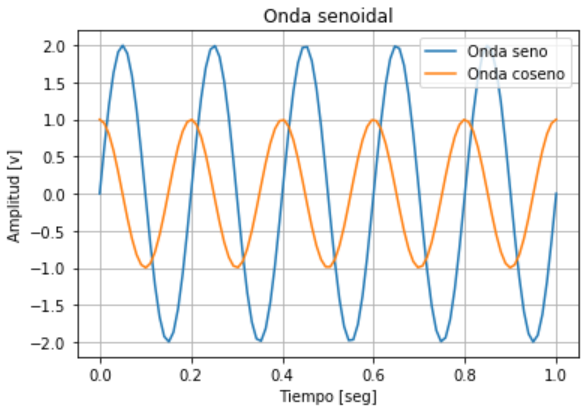

Funcion Senoidal
Sin(x)
Angulo [rad]


exp(x)
Ln(x)

In [28]:

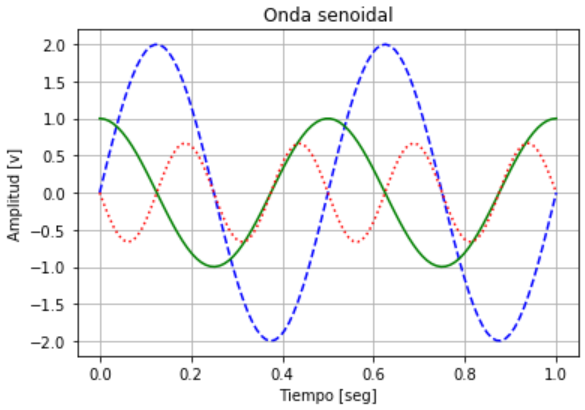
```
# Varias graficas senoidales en una sola ventana

A, f = 2, 5   #Amplitud =5, Frecuencia = 5Hz
t = np.linspace(0, 1, 100)
y1 = A * np.sin(2 * np.pi * f * t)
y2 = A/2 * np.cos(2 * np.pi * f * t)

plt.plot(t, y1, t, y2)
plt.title("Onda senoidal")
plt.xlabel("Tiempo [seg]")
plt.ylabel("Amplitud [v]")
plt.grid()
plt.legend(["Onda seno", "Onda coseno"])
#plt.show()
```

```
<matplotlib.legend.Legend at 0x1afc0a83d30>
```



In [29]:

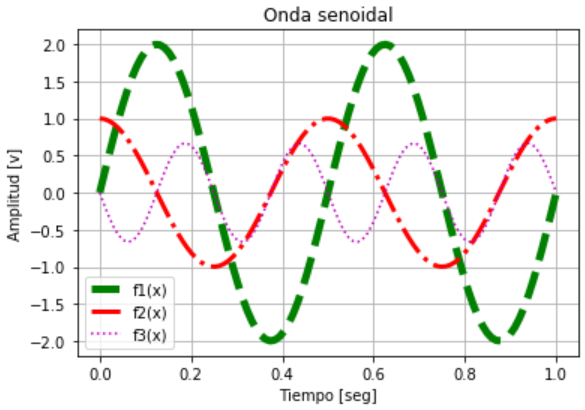
```
#%%
# Ajuste de linea

A, f = 2, 2   #Amplitud =2, Frecuencia = 5Hz
t = np.linspace(0, 1, 100)
y1 = A * np.sin(2 * np.pi * f * t)
y2 = A/2 * np.cos(2 * np.pi * f * t)
y3 = A/3 * -np.sin(2 * np.pi * 2*f * t)

plt.plot(t, y1, '--b', t, y2, '-g', t, y3, ':r')
plt.title("Onda senoidal")
plt.xlabel("Tiempo [seg]")
plt.ylabel("Amplitud [v]")
plt.grid()
#plt.show()
```



Otras herramientas gráficas

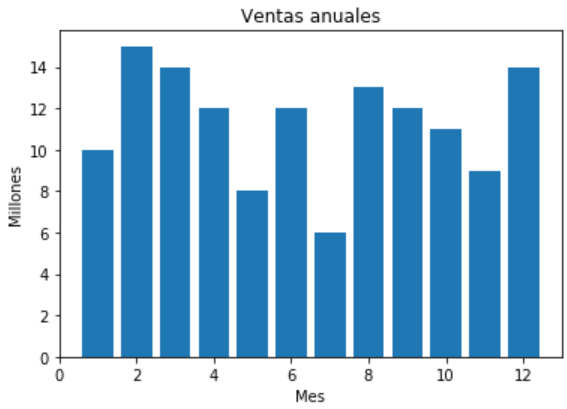
In [10]:

```
# Ajuste de estilo

A, f = 2, 2   #Amplitud =2, Frecuencia = 5Hz
t = np.linspace(0, 1, 100)
y1 = A * np.sin(2 * np.pi * f * t)
y2 = A/2 * np.cos(2 * np.pi * f * t)
y3 = A/3 * -np.sin(2 * np.pi * 2*f * t)

plt.plot(t, y1, linewidth=5, color='g', linestyle='--', label='f1(x)')
plt.plot(t, y2, linewidth=3, color='r', linestyle='-.', label='f2(x)')
plt.plot(t, y3, linewidth=1.5, color='m', linestyle=':', label='f3(x)')
plt.title("Onda senoidal")
plt.xlabel("Tiempo [seg]")
plt.ylabel("Amplitud [v]")
plt.legend(loc='lower left')
plt.grid()
plt.show()
```



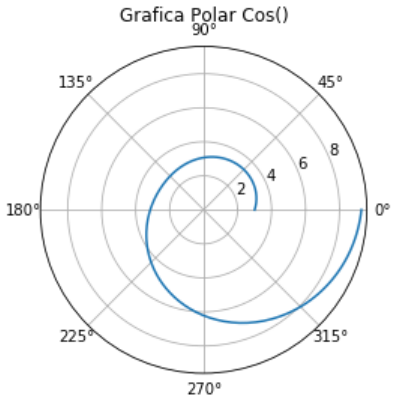
In [21]:

```
# Graficas de barras
ventas = np.array([10, 15, 14, 12, 8, 12, 6, 13, 12, 11, 9, 14])
mes = np.arange(1, 13)
plt.bar(mes, ventas)
plt.title("Ventas anuales")
plt.xlabel("Mes")
plt.ylabel("Millones")
plt.show()
```



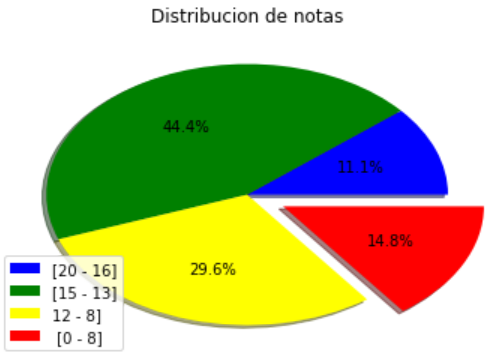
In [22]:

```
# Graficas Polares
t = np.linspace(0, 2*np.pi, 200)
r = 3*np.cos(0.5*t)**2 + t
plt.polar(t, r)
plt.title("Grafica Polar Cos()")
plt.show()
```



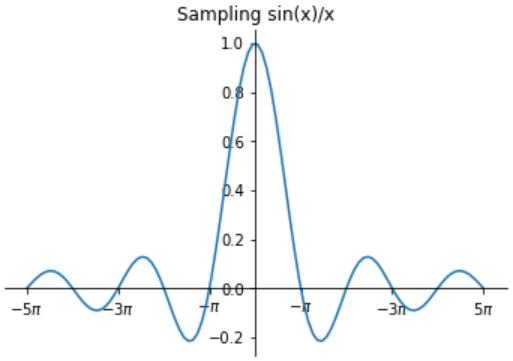
In [23]:

```
# Graficas de tarta
num = [3, 12, 8, 4]
colors = ['blue', 'green', 'yellow', 'red']
explode = [0, 0, 0, 0.2]
plt.pie(num, autopct = '%1.1f%%', explode= explode, colors= colors, shadow= True)
plt.title("Distribucion de notas")
plt.legend(['[20 - 16]', '[15 - 13]', '12 - 8]', ' [0 - 8]'], loc = 'lower left')
plt.show()
```



In [11]:

```
# El objeto gca
x = np.linspace(-5*np.pi, 5*np.pi, 100)
y = np.sin(x) / x

# Se extrae los elementos de los ejes
ax = plt.gca()
# Se elimina el borde superior y derecho
ax.spines['top'].set_color('none')
ax.spines['right'].set_color('none')
# Se mueven los bordes izquierdo e inferior al punto 0
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
# Se modifica los datos de los ejes
plt.xticks([-5*np.pi, -3*np.pi, -np.pi, np.pi, 3*np.pi, 5*np.pi],
           [r'$-5\pi$', r'$-3\pi$', r'$-\pi$', r'$-\pi$', r'$-3\pi$', r'$5\pi$'])
# Se grafica con los ejes ajustados
plt.plot(x,y)
plt.title("Sampling sin(x)/x")
plt.show()
```



EJERCICIOS PROPUESTOS

PROBLEMA 1

Se lanza una bala de cañon con una velocidad inicial de 120 km/h, disparada a un angulo de 60 grados. Obtenga la posicion de la bala de cañon (coordenadas x, y) cada 0.1 segundos hasta que llega al suelo.

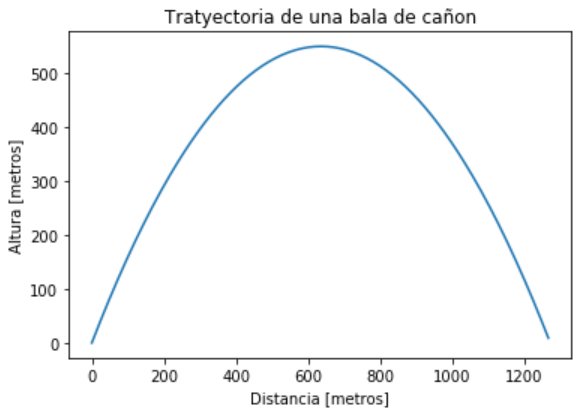
$ t_{vuelo} = 2V.sen(\alpha) / {g} $

$ x = V.cos(\alpha) t $

$ y = V.sen(\alpha) t - \frac{1}{2} g t $

In [28]:

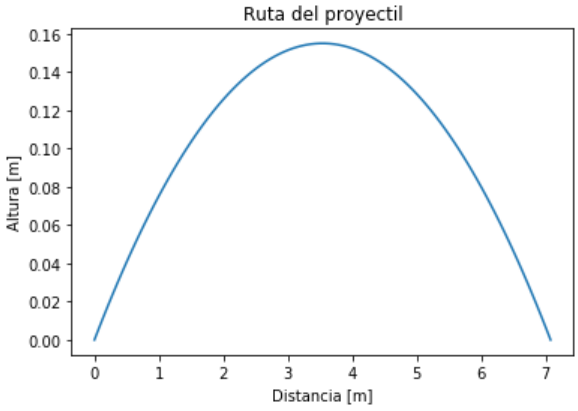
```
import numpy as np
# Datos de entrada
vel = 120; ang =np.radians(60); g = 9.81
# Se calcula el tiempo donde el objeto llega al suelo
tfinal = 2 * vel * np.sin(ang) / g
# Se genera un vector de tiempos
t = np.arange(0, tfinal, 0.1)
# Se halla la posicion x e y de la bala
x = vel * np.cos(ang) * t
y = vel * np.sin(ang) * t - 0.5 * g * t**2
plt.plot(x, y)

# Formato del grafico
plt.title("Tratyectoria de una bala de cañon")
plt.xlabel("Distancia [metros]")
plt.ylabel("Altura [metros]")
plt.show()
```

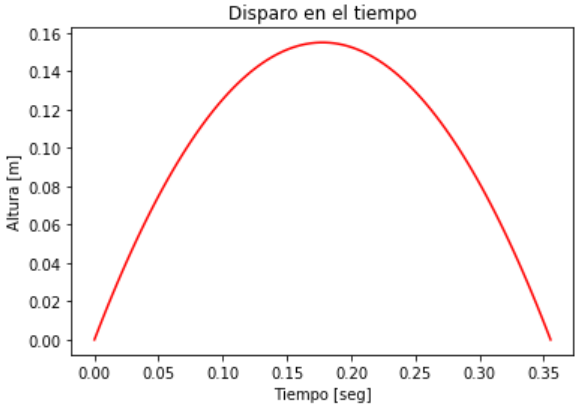


In [20]:

```
# EJERCICIO aplicativo (variacion:
# Script para el calculo de trayectoria balistica
# El modulo numpy tiene todas los metodos matematicos de math, pero con la capacidad
# adicional de que pueden aplicarse a un arreglo
import numpy as np
import matplotlib.pyplot as plt

g = 9.81
Vo = float(input("Ingrese la velocidad de lanzamiento [m/s]: "))
ang = np.radians(float(input("Ingrese el angulo de lanzamiento [grados]: ")))

# Calculo del tiempo de vuelo y la altura maxima
t_vuelo = (2*Vo/g) * np.sin(ang)
x_max = t_vuelo * Vo * np.cos(ang)

# Calculo de la trayectoria
t = np.linspace(0, t_vuelo, 100)   # 100 elementos entre 0 y t_vuelo
x = Vo * np.cos(ang) * t
y = Vo * np.sin(ang) * t - (g/2) * t ** 2

# Calculo de la velocidad y direccion angular del proyectil
vx = Vo* np.cos(ang)
vy = Vo * np.sin(ang) - g * t_vuelo
v = np.sqrt(vx ** 2 + vy ** 2)
ang_v = np.degrees(np.arctan2(vy, vx))

# Calculo del tiempo, distacia horizontal y altitud maxima
tymax = (Vo/g) * np.sin(ang)
xymax = x_max/2
y_max = (Vo/2) * tymax * np.sin(ang)

# Se muestran los resultados
print("\nAlcance: {:.4f} m".format(x_max))
print("Duracion: {:.4f} seg".format(t_vuelo))
print("Altura maxima: {:.4f} m".format(y_max))
print("Llegada: {:.4f} seg".format(tymax))

# Grafico de os resultados
plt.figure(1)
plt.plot(x, y)
plt.title("Ruta del proyectil")
plt.xlabel("Distancia [m]")
plt.ylabel("Altura [m]")
plt.show()

plt.figure(2)
plt.plot(t, y, '-r')
plt.title("Disparo en el tiempo")
plt.xlabel("Tiempo [seg]")
plt.ylabel("Altura [m]")
plt.show()
```

```
Ingrese la velocidad de lanzamiento [m/s]: 20
Ingrese el angulo de lanzamiento [grados]: 5

Alcance: 7.0805 m
Duracion: 0.3554 seg
Altura maxima: 0.1549 m
Llegada: 0.1777 seg
```





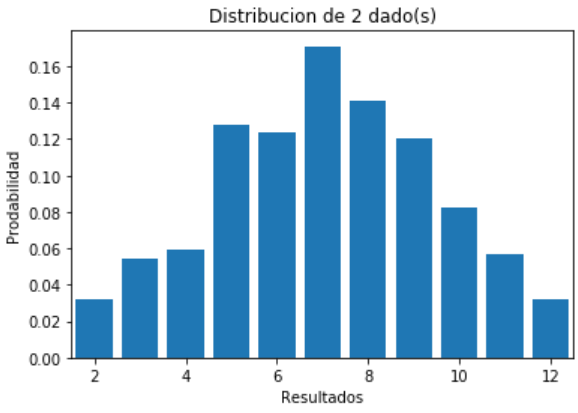
PROBLEMA 2

Analizar la distribución de probabilidad del lanzamiento de *n* dados y para un numero de veces dado.

In [18]:

```
%matplotlib inline
import numpy as np
from matplotlib import pyplot as plt

n_dados = int(input("Ingrese el numero de dados a lanzar: "))
n_veces = int(input("Ingrese el numero de lanzamientos: "))

# Generar un arreglo de jugadas[n_dados, n_veces] con valores entre 1 - 6:
jugadas = np.random.randint(1, 7, (n_dados, n_veces))

# En caso n_dados > 1, voy a sumar las caras de los dados
# para saber el resultado de las jugadas
jugadas = np.sum(jugadas, axis = 0)

# Se genera un arreglo que contenga la distribucion de probabilidad
# Se crea una arreglo inicial con P(cara dado) = 0
prob = np.array(list([0] * (6 * n_dados)))

for i in range(jugadas.size):
    prob[jugadas[i]-1] += 1

prob = prob / n_veces

# Se muestra el grafico con la distribucion de probabilidades
plt.bar(np.arange(1, prob.size+1), prob)
plt.xlim((n_dados - 0.5, 6 * n_dados + 0.5))
plt.title("Distribucion de " + str(n_dados) + " dado(s)")
plt.xlabel("Resultados")
plt.ylabel("Prodabilidad")
plt.show()
```

```
Ingrese el numero de dados a lanzar: 2
Ingrese el numero de lanzamientos: 1000
```



In [ ]: