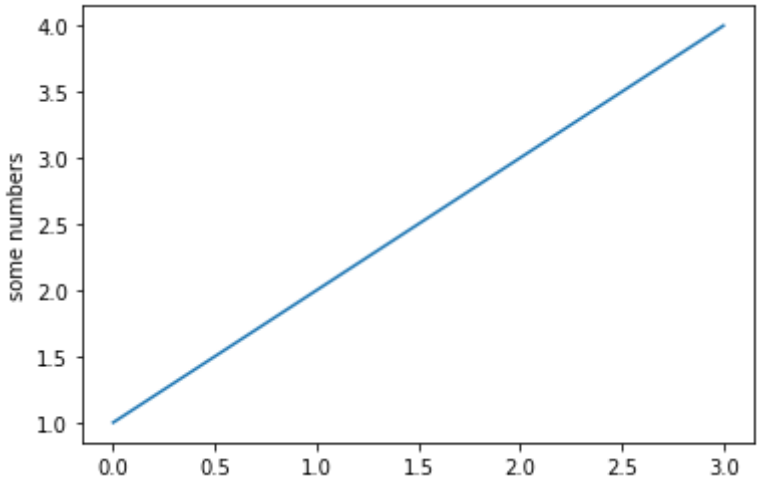
Line Chart

Adding Line to Figures

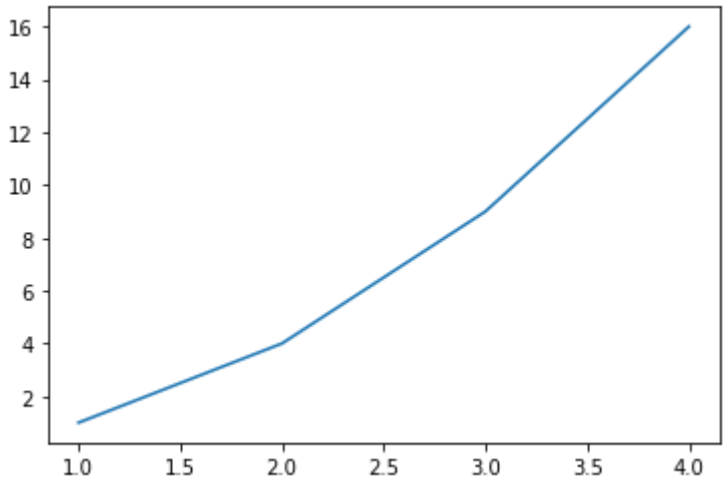
In [3]:
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



Anda mungkin bertanya-tanya mengapa sumbu x berkisar dari 0-3 dan sumbu y dari 1-4. Jika Anda memberikan satu daftar atau larik untuk diplot, matplotlib menganggapnya sebagai urutan nilai y, dan secara otomatis menghasilkan nilai x untuk Anda. Karena rentang python dimulai dengan 0, vektor x default memiliki panjang yang sama dengan y tetapi dimulai dengan 0. Oleh karena itu data x adalah [0, 1, 2, 3].

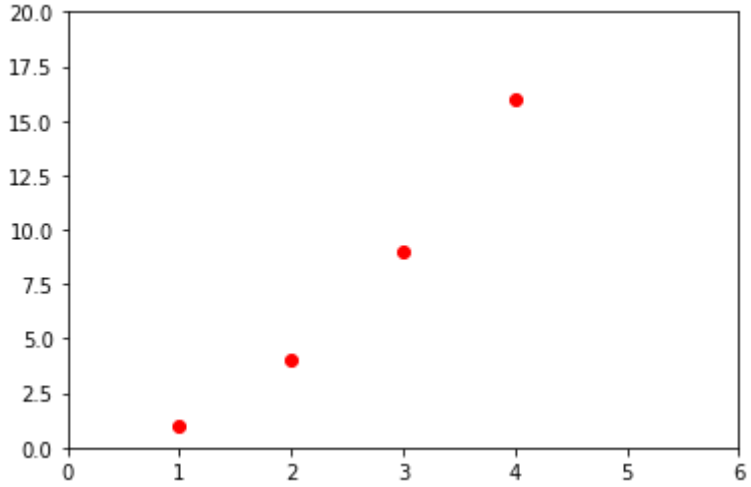
plot adalah fungsi serbaguna, dan akan mengambil sejumlah argumen yang berubah-ubah. Misalnya, untuk memplot x versus y, Anda dapat menulis:

In [5]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.show()
```



Memformat gaya plot Anda Untuk setiap pasangan argumen x, y, ada argumen ketiga opsional yang merupakan format string yang menunjukkan warna dan tipe garis plot. Huruf dan simbol string format berasal dari MATLAB, dan Anda menggabungkan string warna dengan string gaya garis. String format default adalah 'b-', yang merupakan garis biru solid. Misalnya, untuk memplot di atas dengan lingkaran merah, Anda akan mengeluarkan

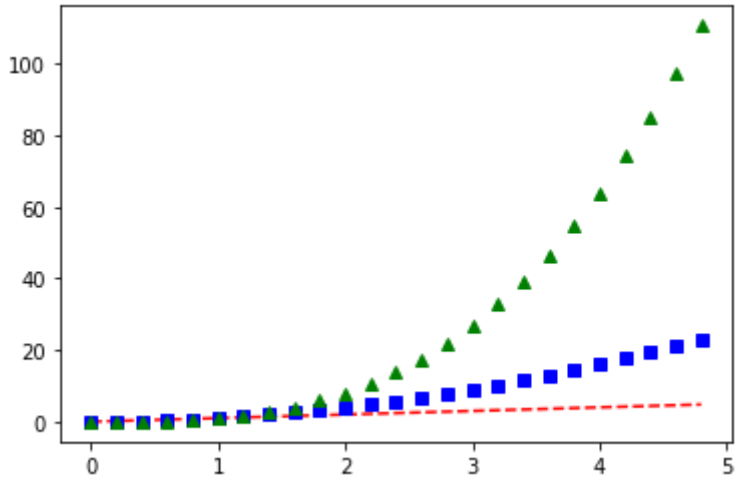
In [6]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



Lihat dokumentasi plot untuk daftar lengkap gaya garis dan format string. Fungsi sumbu dalam contoh di atas mengambil daftar [xmin, xmax, ymin, ymax] dan menentukan area pandang sumbu.

Jika matplotlib terbatas untuk bekerja dengan daftar, itu akan sangat tidak berguna untuk pemrosesan numerik. Umumnya, Anda akan menggunakan array numpy. Faktanya, semua urutan dikonversi ke array numpy secara internal. Contoh di bawah ini menggambarkan plot beberapa baris dengan gaya format yang berbeda dalam satu panggilan fungsi menggunakan array.

In [7]:
```
import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

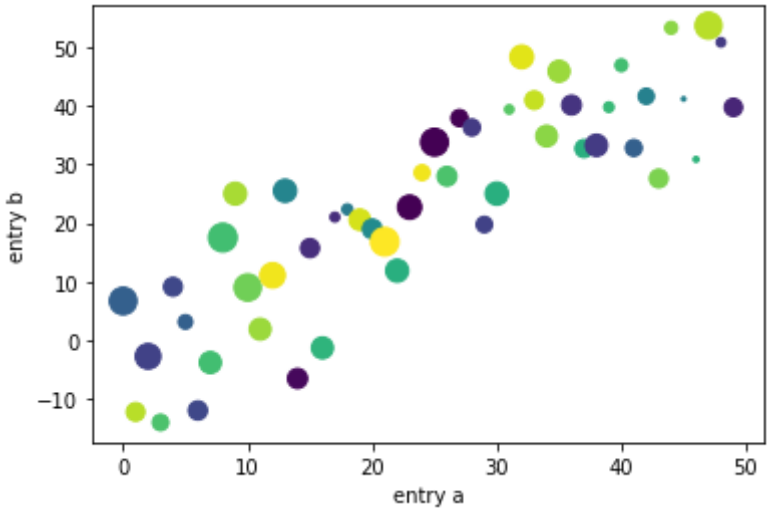


Merencanakan dengan keyword string

Ada beberapa contoh di mana Anda memiliki data dalam format yang memungkinkan Anda mengakses variabel tertentu dengan string. Misalnya, dengan numpy.recarray atau pandas.DataFrame.

Matplotlib memungkinkan Anda menyediakan objek seperti itu dengan argumen kata kunci data. Jika disediakan, maka Anda dapat membuat plot dengan string yang sesuai dengan variabel ini.

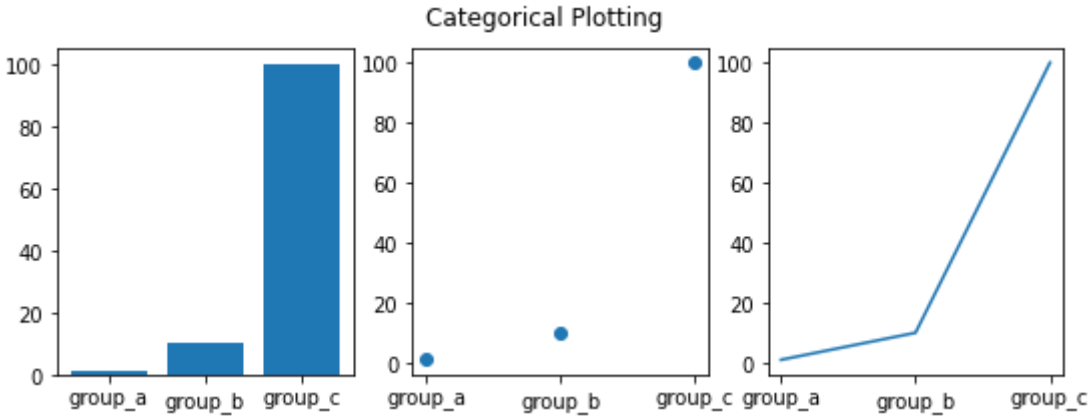
In [8]:
```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()
```



Memplot dengan variabel kategori

Dimungkinkan juga untuk membuat plot menggunakan variabel kategori. Matplotlib memungkinkan Anda untuk meneruskan variabel kategorikal secara langsung ke banyak fungsi plot. Sebagai contoh:

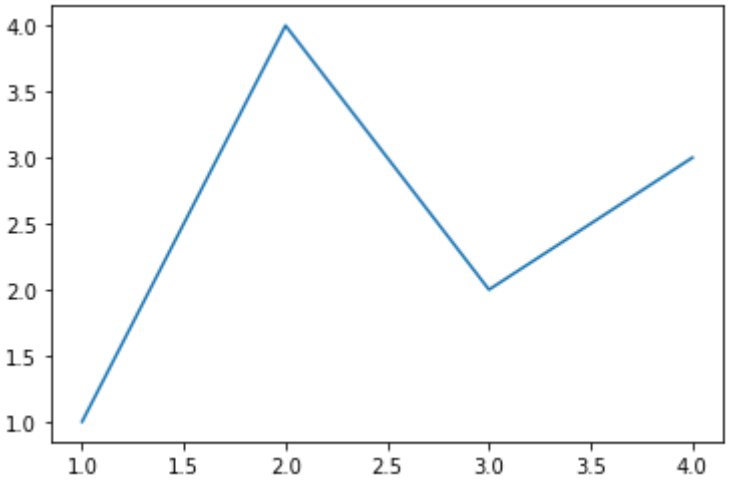
In [9]:
```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]

plt.figure(figsize=(9, 3))

plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```



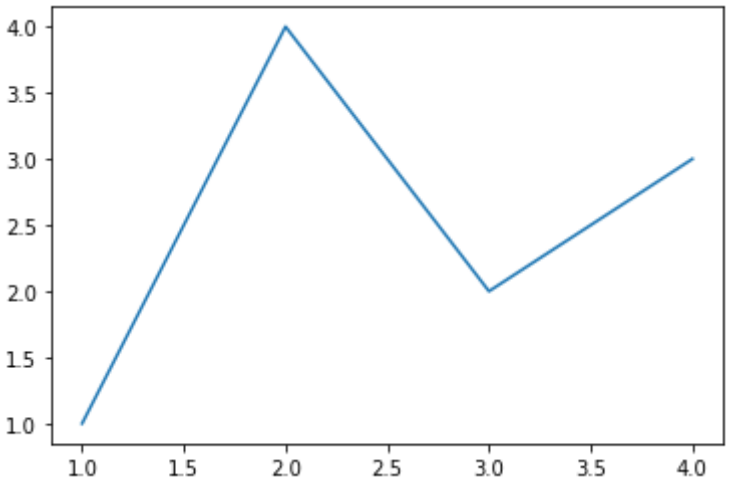
Contoh sederhana

Cara paling sederhana untuk membuat gambar dengan sumbu adalah menggunakan pyplot.subplots. Kami kemudian dapat menggunakan Axes.plot untuk menggambar beberapa data pada sumbu:

In [11]:
```
fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Plot some data on the axes.
plt.show()
```



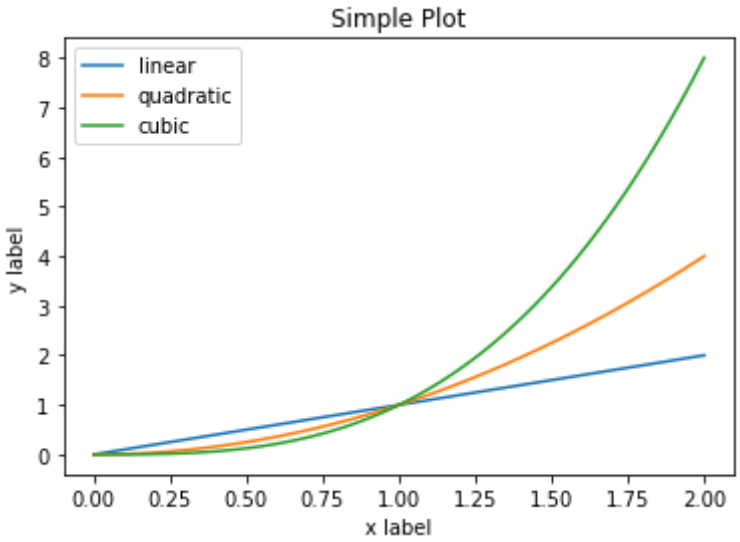
In [13]:
```
plt.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Matplotlib plot.
plt.show()
```



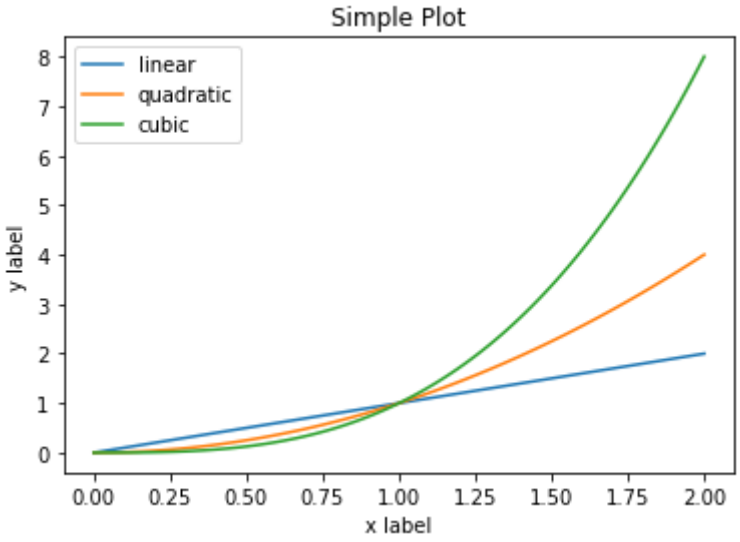
Interface berorientasi objek dan antarmuka pyplot

Seperti disebutkan di atas, pada dasarnya ada dua cara untuk menggunakan Matplotlib: Buat gambar dan sumbu secara eksplisit, dan panggil metode padanya OOP Andalkan pyplot untuk membuat dan mengelola gambar dan sumbu secara otomatis, dan menggunakan fungsi pyplot untuk membuat plot.

In [15]:
```
x = np.linspace(0, 2, 100)

# Note that even in the OO-style, we use `.pyplot.figure` to create the figu
fig, ax = plt.subplots()  # Create a figure and an axes.
ax.plot(x, x, label='linear')  # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
ax.plot(x, x**3, label='cubic')  # ... and some more.
ax.set_xlabel('x label')  # Add an x-label to the axes.
ax.set_ylabel('y label')  # Add a y-label to the axes.
ax.set_title("Simple Plot")  # Add a title to the axes.
ax.legend()  # Add a legend.
plt.show()
```

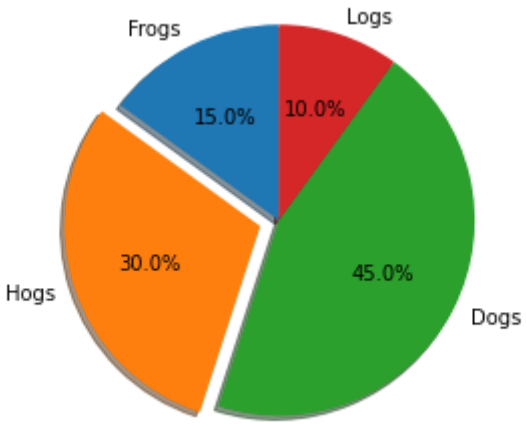


In [17]:
```
x = np.linspace(0, 2, 100)

plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.
plt.plot(x, x**2, label='quadratic')  # etc.
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
plt.show()
```



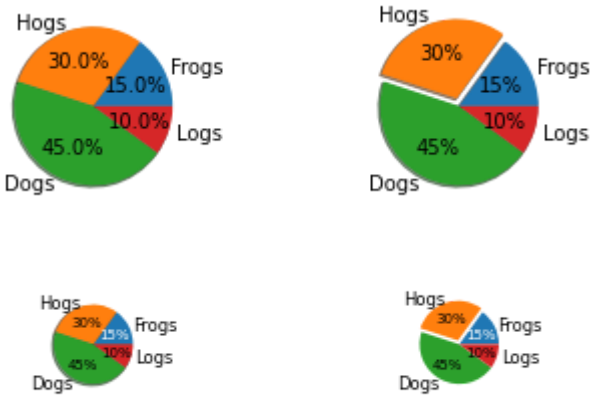
Pie Chart

In [18]:
```
import matplotlib.pyplot as plt

# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle

plt.show()
```



In [19]:
```
import matplotlib.pyplot as plt

# Some data
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
fracs = [15, 30, 45, 10]

# Make figure and axes
fig, axs = plt.subplots(2, 2)

# A standard pie plot
axs[0, 0].pie(fracs, labels=labels, autopct='%1.1f%%', shadow=True)

# Shift the second slice using explode
axs[0, 1].pie(fracs, labels=labels, autopct='%.0f%%', shadow=True,
```

```
                explode=(0, 0.1, 0, 0))

# Adapt radius and text size for a smaller pie
patches, texts, autotexts = axs[1, 0].pie(fracs, labels=labels,
                                          autopct='%.0f%%',
                                          textprops={'size': 'smaller'},
                                          shadow=True, radius=0.5)
# Make percent texts even smaller
plt.setp(autotexts, size='x-small')
autotexts[0].set_color('white')

# Use a smaller explode and turn of the shadow for better visibility
patches, texts, autotexts = axs[1, 1].pie(fracs, labels=labels,
                                          autopct='%.0f%%',
                                          textprops={'size': 'smaller'},
                                          shadow=False, radius=0.5,
                                          explode=(0, 0.05, 0, 0))
plt.setp(autotexts, size='x-small')
autotexts[0].set_color('white')

plt.show()
```



Nested Pie Chart

In [20]:

```
fig, ax = plt.subplots()

size = 0.3
vals = np.array([[60., 32.], [37., 40.], [29., 10.]])

cmap = plt.get_cmap("tab20c")
outer_colors = cmap(np.arange(3)*4)
inner_colors = cmap([1, 2, 5, 6, 9, 10])

ax.pie(vals.sum(axis=1), radius=1, colors=outer_colors,
       wedgeprops=dict(width=size, edgecolor='w'))

ax.pie(vals.flatten(), radius=1-size, colors=inner_colors,
       wedgeprops=dict(width=size, edgecolor='w'))

ax.set(aspect="equal", title='Pie plot with `ax.pie`')
plt.show()
```

Pie plot with `ax.pie`

In [21]:

```
fig, ax = plt.subplots(subplot_kw=dict(projection="polar"))

size = 0.3
vals = np.array([[60., 32.], [37., 40.], [29., 10.]])
#normalize vals to 2 pi
valsnorm = vals/np.sum(vals)*2*np.pi
#obtain the ordinates of the bar edges
valsleft = np.cumsum(np.append(0, valsnorm.flatten()[:-1])).reshape(vals.shap

cmap = plt.get_cmap("tab20c")
outer_colors = cmap(np.arange(3)*4)
inner_colors = cmap([1, 2, 5, 6, 9, 10])

ax.bar(x=valsleft[:, 0],
       width=valsnorm.sum(axis=1), bottom=1-size, height=size,
       color=outer_colors, edgecolor='w', linewidth=1, align="edge")

ax.bar(x=valsleft.flatten(),
       width=valsnorm.flatten(), bottom=1-2*size, height=size,
       color=inner_colors, edgecolor='w', linewidth=1, align="edge")

ax.set(title="Pie plot with `ax.bar` and polar coordinates")
ax.set_axis_off()
plt.show()
```

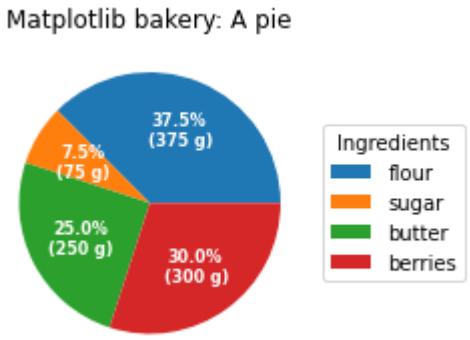
Pie plot with `ax.bar` and polar coordinates

Melabel Pie dan Donut Chart

In [22]:

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

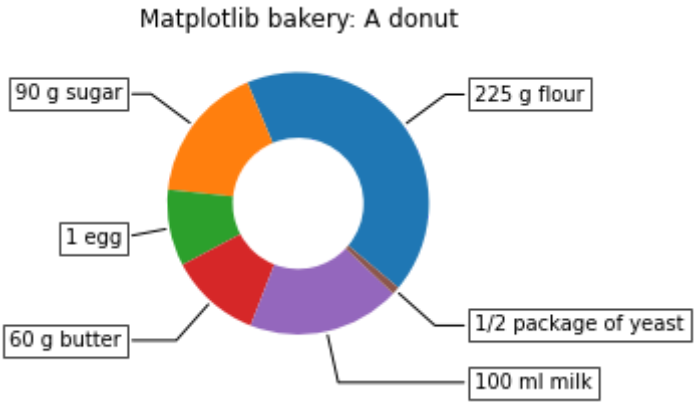
```
fig, ax = plt.subplots(figsize=(6, 3), subplot_kw=dict(aspect="equal"))

recipe = ["375 g flour",
          "75 g sugar",
          "250 g butter",
          "300 g berries"]

data = [float(x.split()[0]) for x in recipe]
ingredients = [x.split()[-1] for x in recipe]


def func(pct, allvals):
    absolute = int(round(pct/100.*np.sum(allvals)))
    return "{:.1f}%\n({:d} g)".format(pct, absolute)


wedges, texts, autotexts = ax.pie(data, autopct=lambda pct: func(pct, data),
                                  textprops=dict(color="w"))

ax.legend(wedges, ingredients,
          title="Ingredients",
          loc="center left",
          bbox_to_anchor=(1, 0, 0.5, 1))

plt.setp(autotexts, size=8, weight="bold")

ax.set_title("Matplotlib bakery: A pie")

plt.show()
```



In [23]:

```
fig, ax = plt.subplots(figsize=(6, 3), subplot_kw=dict(aspect="equal"))

recipe = ["225 g flour",
          "90 g sugar",
          "1 egg",
          "60 g butter",
          "100 ml milk",
          "1/2 package of yeast"]

data = [225, 90, 50, 60, 100, 5]

wedges, texts = ax.pie(data, wedgeprops=dict(width=0.5), startangle=-40)

bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
kw = dict(arrowprops=dict(arrowstyle="-"),
          bbox=bbox_props, zorder=0, va="center")

for i, p in enumerate(wedges):
    ang = (p.theta2 - p.theta1)/2. + p.theta1
    y = np.sin(np.deg2rad(ang))
```

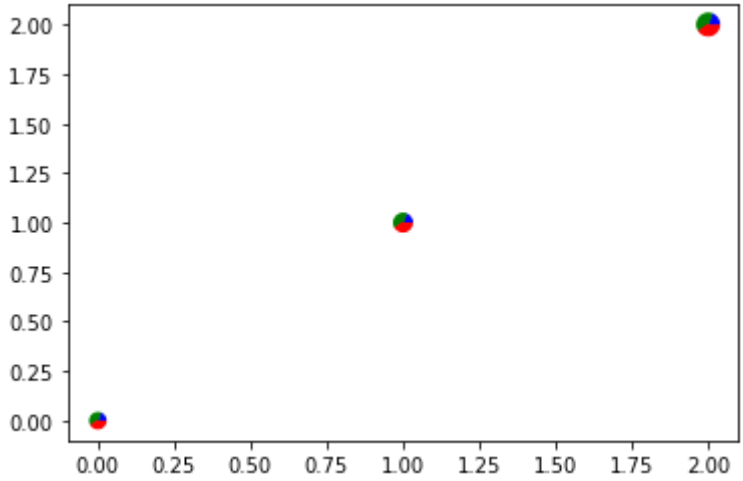
```
    x = np.cos(np.deg2rad(ang))
    horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
    connectionstyle = "angle,angleA=0,angleB={}".format(ang)
    kw["arrowprops"].update({"connectionstyle": connectionstyle})
    ax.annotate(recipe[i], xy=(x, y), xytext=(1.35*np.sign(x), 1.4*y),
                horizontalalignment=horizontalalignment, **kw)

ax.set_title("Matplotlib bakery: A donut")

plt.show()
```



Scatter Plot dengan Marker nya Pie

In [24]:

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

# first define the ratios
r1 = 0.2       # 20%
r2 = r1 + 0.4  # 40%

# define some sizes of the scatter marker
sizes = np.array([60, 80, 120])

# calculate the points of the first pie marker
# these are just the origin (0, 0) + some (cos, sin) points on a circle
x1 = np.cos(2 * np.pi * np.linspace(0, r1))
y1 = np.sin(2 * np.pi * np.linspace(0, r1))
xy1 = np.row_stack([[0, 0], np.column_stack([x1, y1])])
s1 = np.abs(xy1).max()

x2 = np.cos(2 * np.pi * np.linspace(r1, r2))
y2 = np.sin(2 * np.pi * np.linspace(r1, r2))
xy2 = np.row_stack([[0, 0], np.column_stack([x2, y2])])
s2 = np.abs(xy2).max()

x3 = np.cos(2 * np.pi * np.linspace(r2, 1))
y3 = np.sin(2 * np.pi * np.linspace(r2, 1))
xy3 = np.row_stack([[0, 0], np.column_stack([x3, y3])])
s3 = np.abs(xy3).max()

fig, ax = plt.subplots()
ax.scatter(range(3), range(3), marker=xy1, s=s1**2 * sizes, facecolor='blue')
ax.scatter(range(3), range(3), marker=xy2, s=s2**2 * sizes, facecolor='green'
ax.scatter(range(3), range(3), marker=xy3, s=s3**2 * sizes, facecolor='red')

plt.show()
```


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

In [ ]: