

Matlibplot入門

前回の復習

課題 簡易マスターメンテ

仕様追加(matrtmente1.pyに追加)



認証画面

UID

PWD

検索 追加 クリア 削除

- 1,削除機能をつけてください
- 2,uidがないときや削除したときメッセージを出してください

```
btn3 = tk.Button(root, text='クリア', command=btn_click3)
btn3.place(x=170, y=160) #表示位置
```

```
btn4 = tk.Button(root, text='削除', command=btn_click4)
btn4.place(x=210, y=160) #表示位置
```

```
lblmsg = tk.Label(text=u" ") #
lblmsg.place(x=30, y=180)
```

btn3.placeの下に追加する
ボタンとラベル

btn_click4を実装

#クリアボタン

```
def btn_click3():  
    txtuid.delete(0,tk.END)  
    txtpwd.delete(0,tk.END)
```

クリアボタンの後ろに削除ボタンの処理を書く

#削除ボタン

```
def btn_click4():  
    uid=txtuid.get()  
    print(uid)  
    delete(uid)
```

delete関数を作り実装

delete関数を実装

```
#-----  
#delete  
#引数 uid  
#-----  
def delete(uid):  
    dbname='TestDB.db'  
    conn=sqlite3.connect(dbname)  
    c = conn.cursor()  
    delete_sql =XXXXXXXXXXXXXXXXXXXXX  
    print("SQL=",delete_sql)  
    c.execute(XXXXXXX)  
    conn.commit()  
    conn.close()  
    #テキストボックスクリア  
    txtuid.delete(0,tk.END)  
    txtpwd.delete(0,tk.END)  
    lblmsg['text']="削除しました"
```

lblmsgのラベルを書き換えるときは
オブジェクト名['text']="文字列"

kensaku関数にメッセージを出す

```
def kensaku(uid):  
    dbname='TestDB.db'  
    conn=sqlite3.connect(dbname)  
    select_sql = "select uid,pwd from user where  
uid='"+uid+"'"  
    print(select_sql)  
    flg=0  
    c = conn.cursor()  
    for row in c.execute(select_sql):  
        print(row[0],"*****",row[1])  
        print("OK")  
        uid=row[0]  
        pwd=row[1]  
        txtpwd.insert(tk.END,pwd)  
        flg=1  
    conn.close()  
    if flg==0:  
        print("データない")  
        XXXXXX="データない"
```

データがないときメッセージをだす

解答例 `mastrtmente3.py`

折れ線グラフ

パラメータ

- title,xlabel,ylabel,grid,xtick,legend,figure
- addsubplot,subplots_adjust,set_xlim,set_ylim
set_xlabel,set_ylabel

折れ線グラフ例

```
%matplotlib inline
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], # xの値
         [1, 4, 9, 16])# yの値
plt.ylabel('y-label') # y軸のラベルをプロット
plt.xlabel('x-label') # x軸のラベルをプロット
plt.show()
```

折れ線グラフを表示する(plot,show)

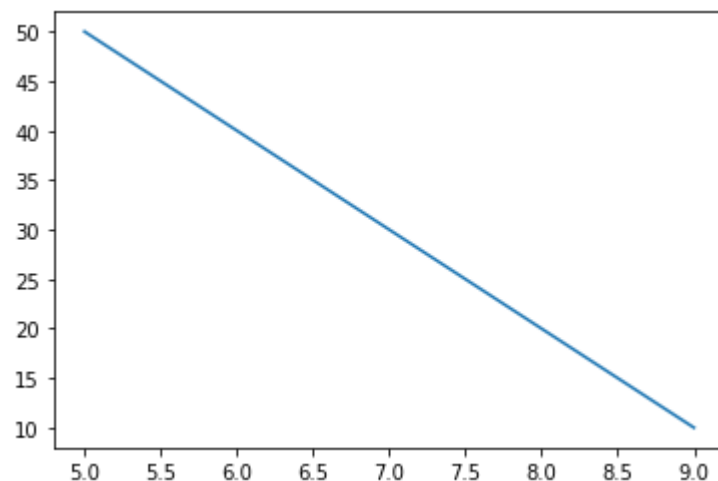
```
import matplotlib.pyplot as plt
```

```
x_list = [ 5, 6, 7, 8, 9]
```

```
y_list = [50, 40, 30, 20, 10]
```

```
plt.plot(x_list, y_list)
```

```
plt.show()
```



グラフのタイトル(plot2.py)(title)

```
import matplotlib.pyplot as plt
```

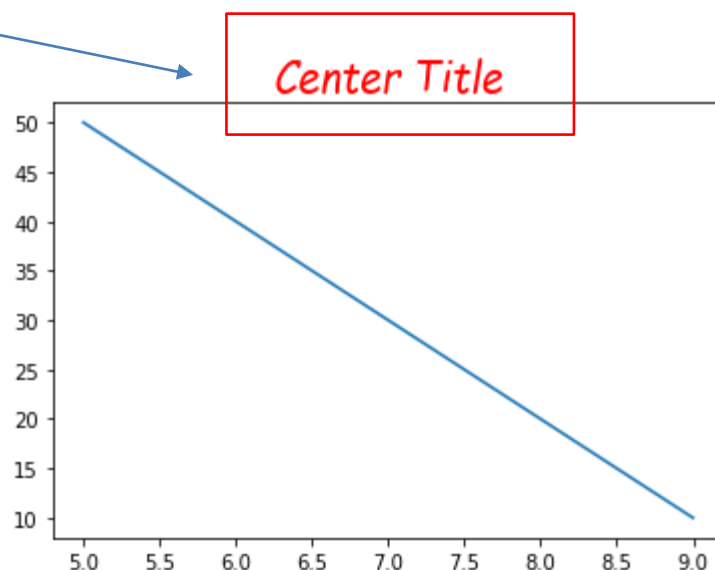
```
x_list = [ 5, 6, 7, 8, 9]
```

```
y_list = [50, 40, 30, 20, 10]
```

```
plt.title('Center Title',  
          color='red',  
          size=20,  
          family='fantasy',  
          )
```

```
plt.plot(x_list, y_list)
```

```
plt.show()
```



x軸y軸のタイトル

```
import matplotlib.pyplot as plt
```

```
x_list = [ 5, 6, 7, 8, 9]
```

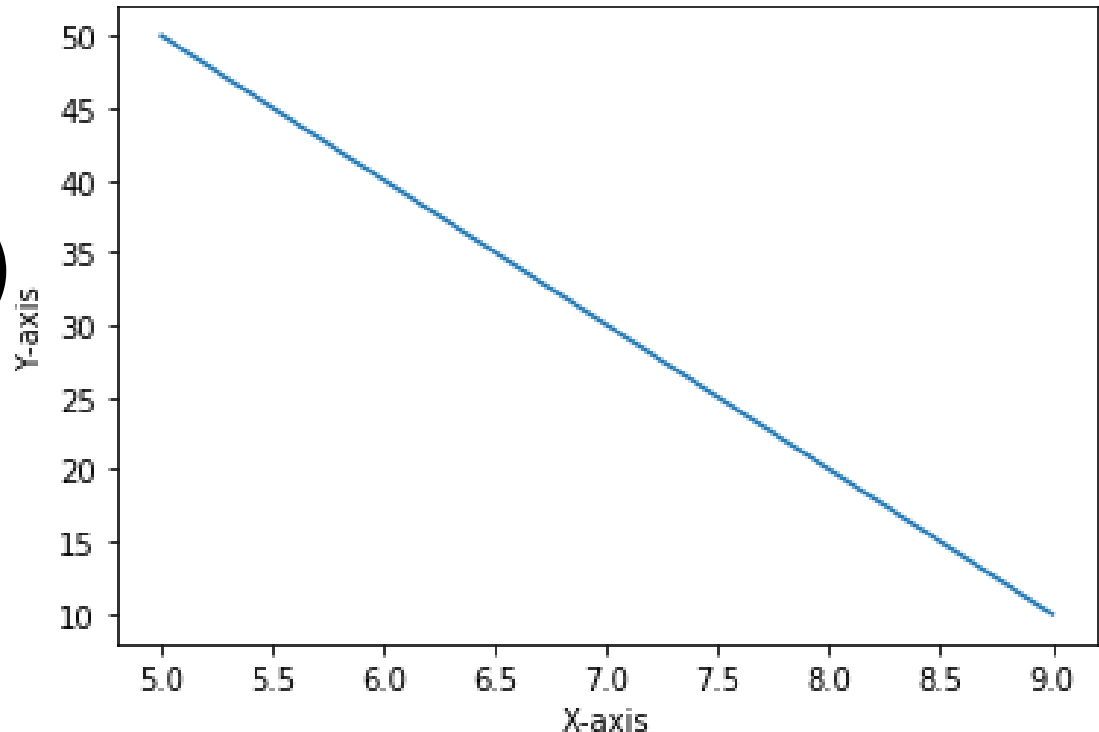
```
y_list = [50, 40, 30, 20, 10]
```

```
plt.xlabel("X-axis")
```

```
plt.ylabel("Y-axis")
```

```
plt.plot(x_list, y_list)
```

```
plt.show()
```



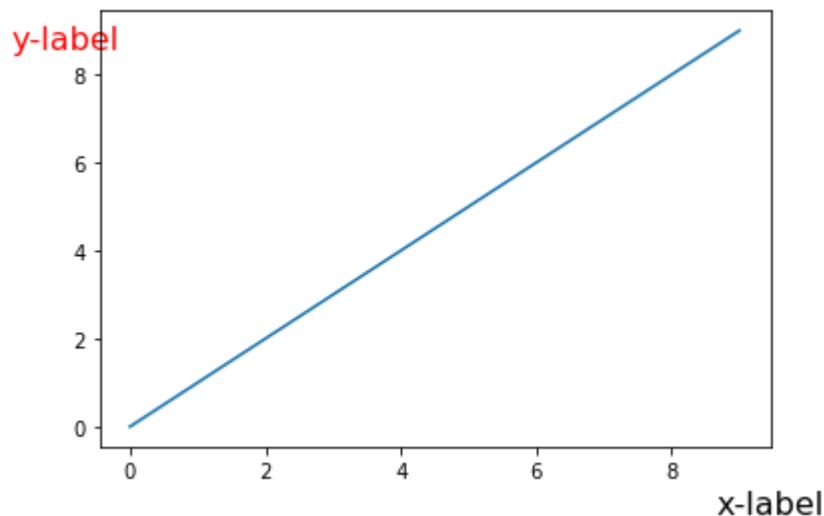
X軸,y軸(xlabel,ylabel)のタイトルのつけ方

```
plt.plot(range(10))      # y値を0～10にしてラインをプロット
```

```
plt.xlabel('x-label',    # x軸ラベルのテキスト
           size=16,
           position=(1,0), # x軸に対して1の位置(右端)に配置
           rotation=0      # テキストの回転角度を0にする
           )
```

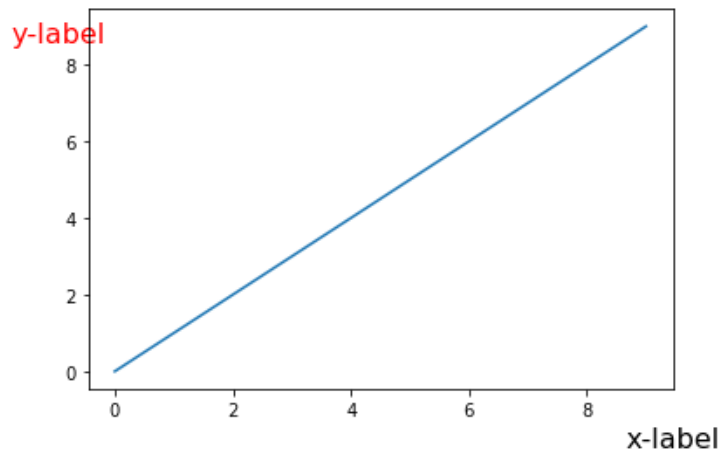
```
plt.ylabel('y-label',    # y軸ラベルのテキスト
           color='red',
           size=16,
           position=(0, 0.9), # y軸に対して0.9の位置に配置
           rotation=0      # テキストの回転角度を0にする
           )
```

```
plt.show()
```

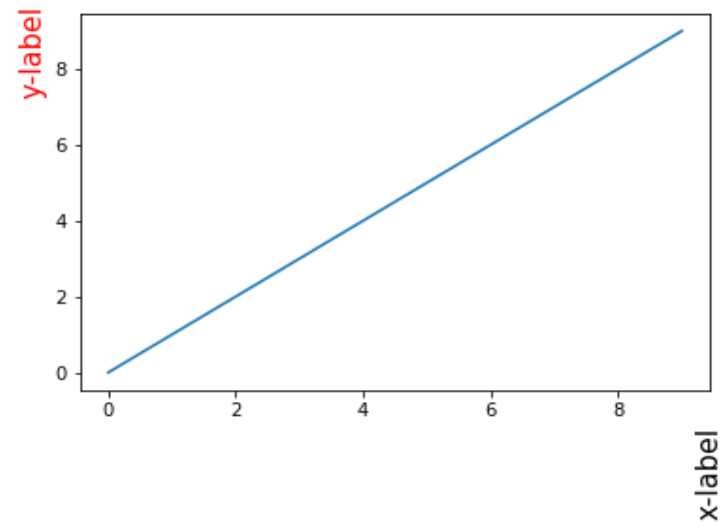


rotationパラメータを変えてみる

rotation=0

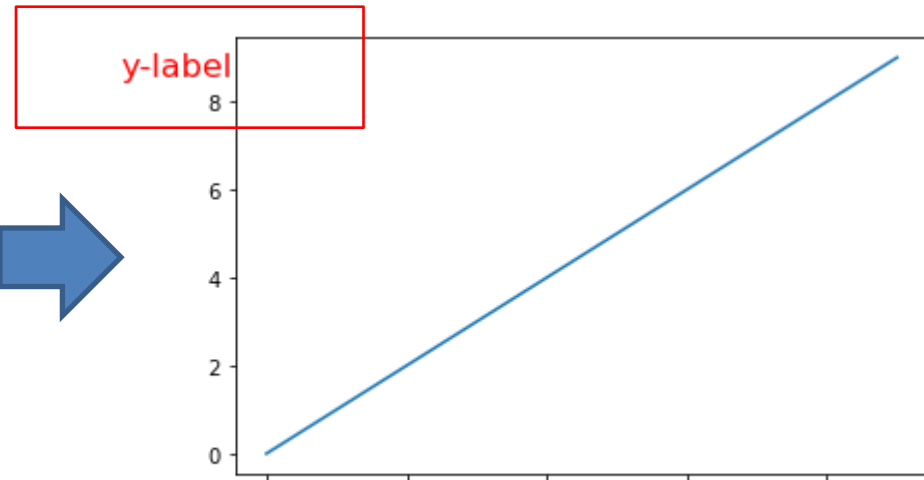
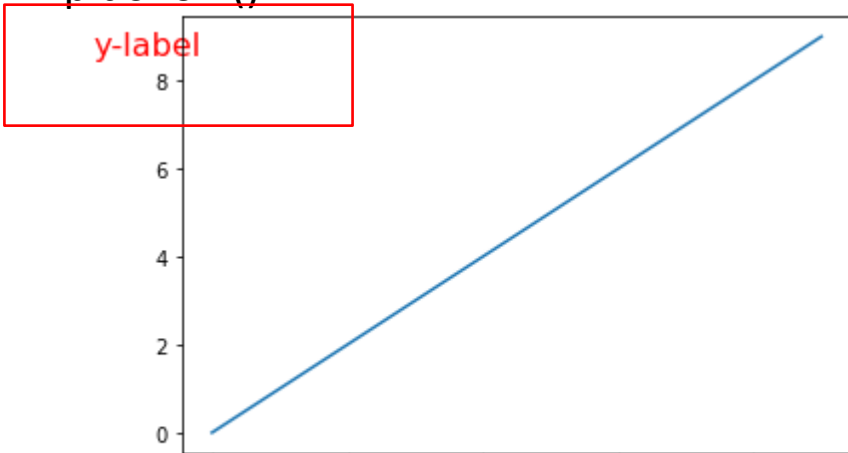


rotation=90



空白を入れてみる

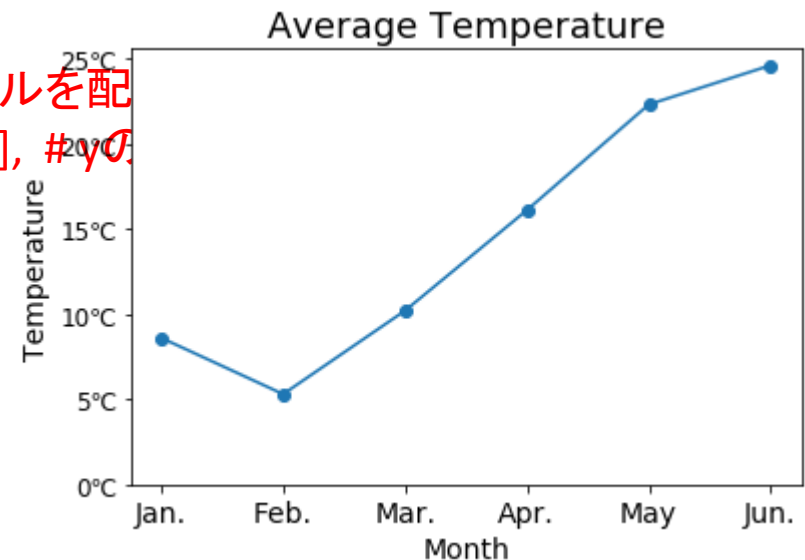
```
plt.plot(range(10))          # y値を0～10にしてラインをプロット
plt.xlabel('x-label',        # x軸ラベルのテキスト
           size=16,
           position=(1,0),    # x軸に対して1の位置(右端)に配置
           rotation=0         # 90度反時計回りに回転
)
plt.ylabel('y-label',        # y軸ラベルのテキスト
           color='red',
           size=16,
           position=(0, 0.9), # y軸に対して0.9の位置に配置
           labelpad=15,
           rotation=0         # テキストの回転角度を0にする
)
plt.show()
```



軸の目盛りに単位を入れる (xticks,yticks)

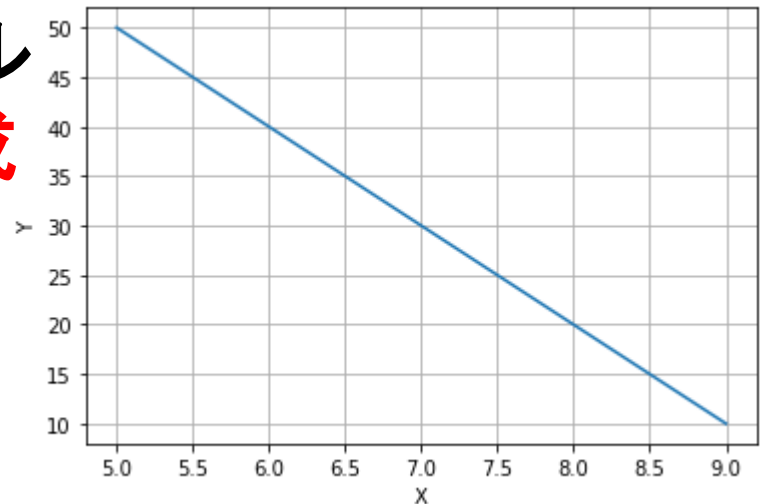
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4, 5, 6],          # xの値 月
         [8.6, 5.3, 10.2, 16.1, 22.3, 24.6], # yの値 気温
         marker='o',                  # サークル型のマーカー
         )
plt.title('Average Temperature', size=18) # タイトル
plt.xlabel('Month', size=14)              # x軸のラベルをプロット
plt.ylabel('Temperature', size=14)        # y軸のラベルをプロット
plt.xticks([1, 2, 3, 4, 5, 6],          # 目盛ラベルを配置するx軸の位置
           ['Jan.', 'Feb.', 'Mar.', 'Apr.', 'May', 'Jun.'], # xの目盛ラベル
           size=14
           )
plt.yticks([0, 5, 10, 15, 20, 25],      # 目盛ラベルを配置するy軸の位置
           ['0°C', '5°C', '10°C', '15°C', '20°C', '25°C'], # yの目盛ラベル
           size=12
           )

plt.show()
```



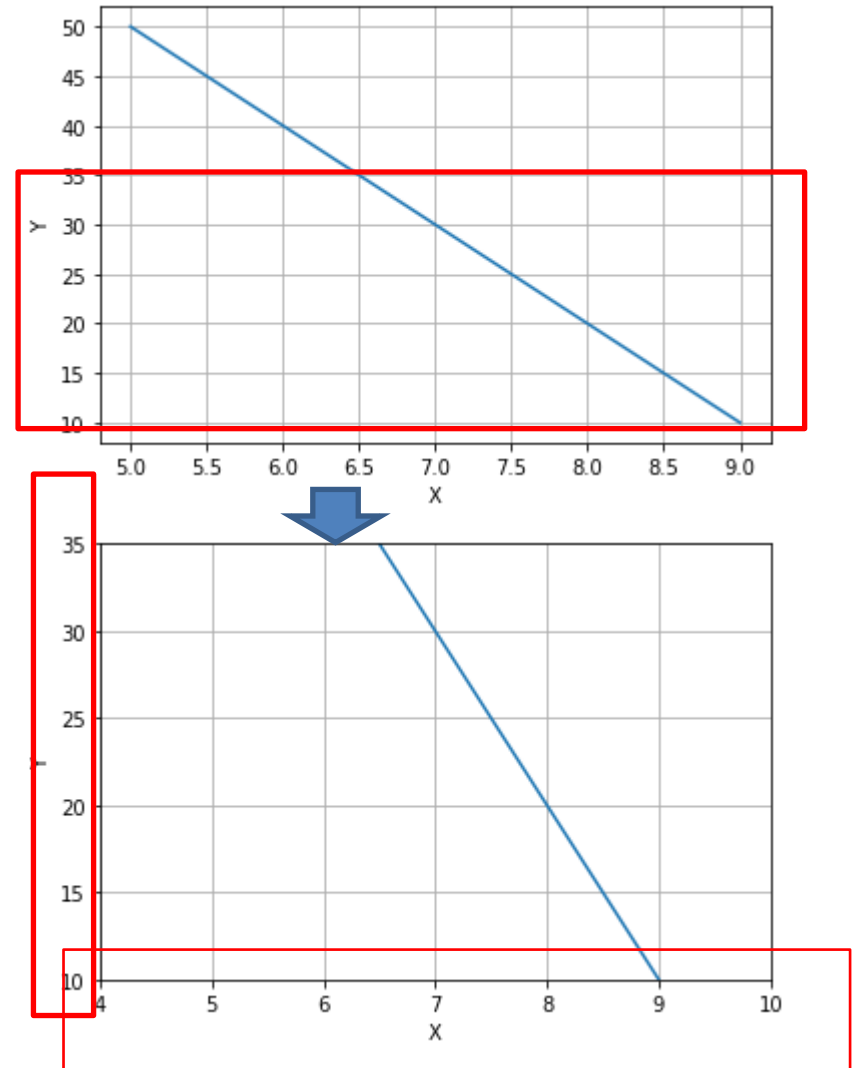
グリッド作成

```
import matplotlib.pyplot as plt
x_list = [ 5, 6, 7, 8, 9]
y_list = [50, 40, 30, 20, 10]
plt.title="Title"
plt.xlabel('X')#x軸のラベル
plt.ylabel('Y')#y軸のラベル
plt.grid()      #グリッド作成
plt.plot(x_list, y_list)
plt.show()
```



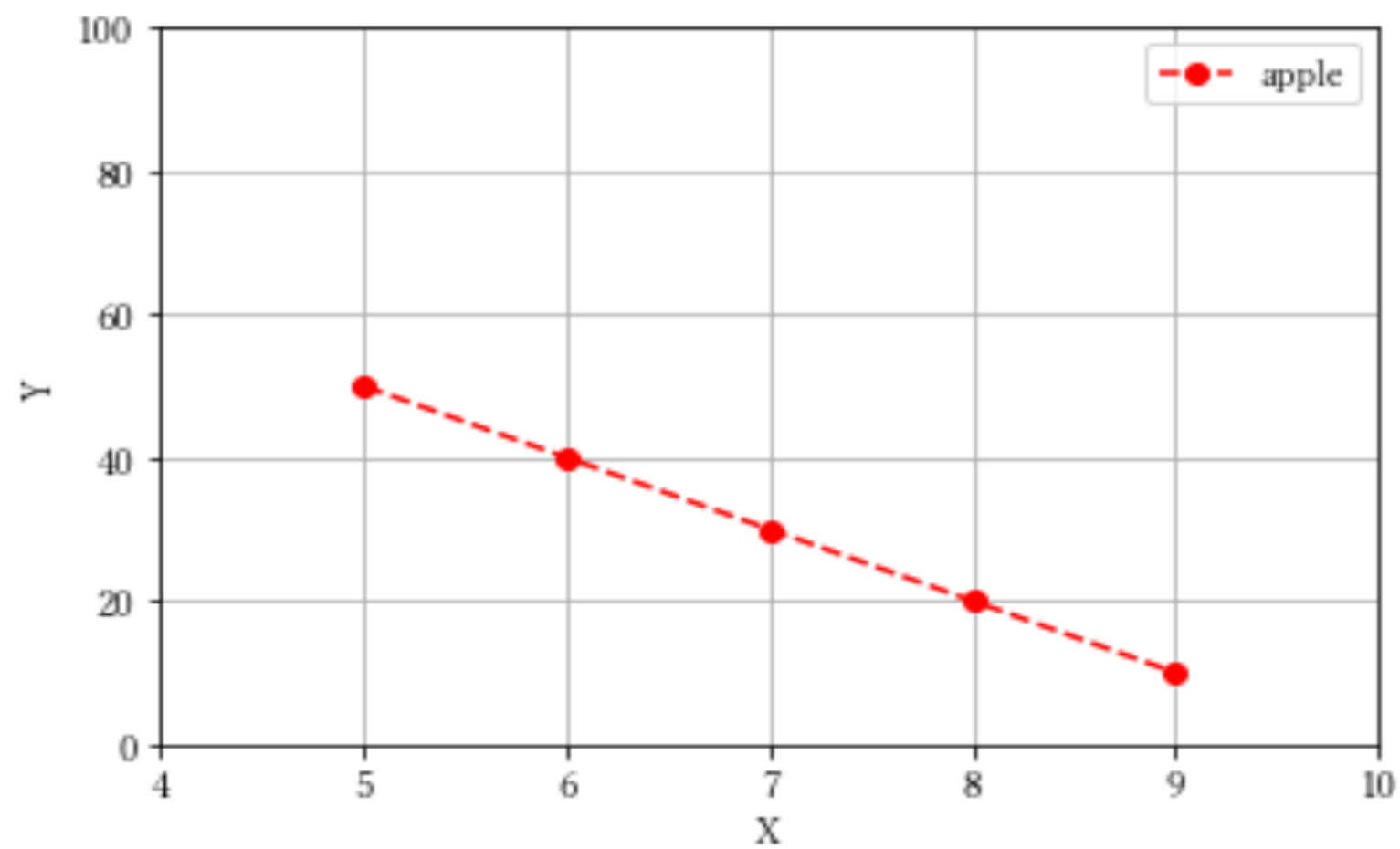
xlim,ylim(表示範囲を制限する)

```
import matplotlib.pyplot as plt
x_list = [ 5, 6, 7, 8, 9]
y_list = [50, 40, 30, 20, 10]
plt.title="Title"
plt.xlim([4,10])
plt.ylim([10,35])
plt.xlabel('X')#x軸のラベル
plt.ylabel('Y')#y軸のラベル
plt.grid()      #グリッド作成
plt.plot(x_list, y_list)
plt.show()
```



マーカー、ライン、色

```
import matplotlib.pyplot as plt
x_list = [ 5, 6, 7, 8, 9]
y_list = [50, 40, 30, 20, 10]
plt.title('Title') # グラフのタイトル
plt.xlabel('X')# X軸のラベル
plt.ylabel('Y')# Y軸のラベル
plt.xlim([4, 10]) # xグラフの表示範囲
plt.ylim([0, 100]) # yグラフの表示範囲
plt.grid() # グリッドの表示
# 書式
marker = 'o'
line = '--'
color = 'r' # b:青 g:緑 r:赤 c:シアン m:マゼンダ y:黄 k:黒 w:白
fmt = marker + line + color
plt.plot(x_list, y_list, fmt, label = 'apple')# グラフデータの設定
plt.legend() # 凡例の表示
plt.show() # グラフの表示
```







Line

line = '--'

line = '-'

line = ':'

line = '-.'

	linestyle : solid, '-'
	linestyle : dashed, '--'
	linestyle : dashdot, '-.'
	linestyle : dotted, ':'

marker

marker	symbol	description
"."	•	point
","	·	pixel
"o"	●	circle
"v"	▼	triangle_down
"^"	▲	triangle_up
"<"	◀	triangle_left
">"	▶	triangle_right
"1"	⤵	tri_down
"2"	⤴	tri_up
"3"	⤶	tri_left
"4"	⤷	tri_right
"8"	⬢	octagon
"s"	■	square
"p"	⬠	pentagon
"P"	⊕	plus (filled)
"*"	★	star

以下のコードでマーカー確かめることができます

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
x = np.arange(1, 11)
```

```
y1 = np.repeat(3, 10) # 3を10回繰り返す
```

```
y2 = np.repeat(2, 10) # 2を10回繰り返す
```

```
y3 = np.repeat(1, 10) # 1を10回繰り返す
```

```
markers1 = [".", ",", "o", "v", "^", "<", ">", "1", "2", "3"]
```

```
markers2 = ["4", "8", "s", "p", "*", "h", "H", "+", "x", "D"]
```

```
markers3 = ["d", "|", "_", "None", None, "",  
"$x$", "$\alpha$", "$\beta$", "$\gamma$"]
```

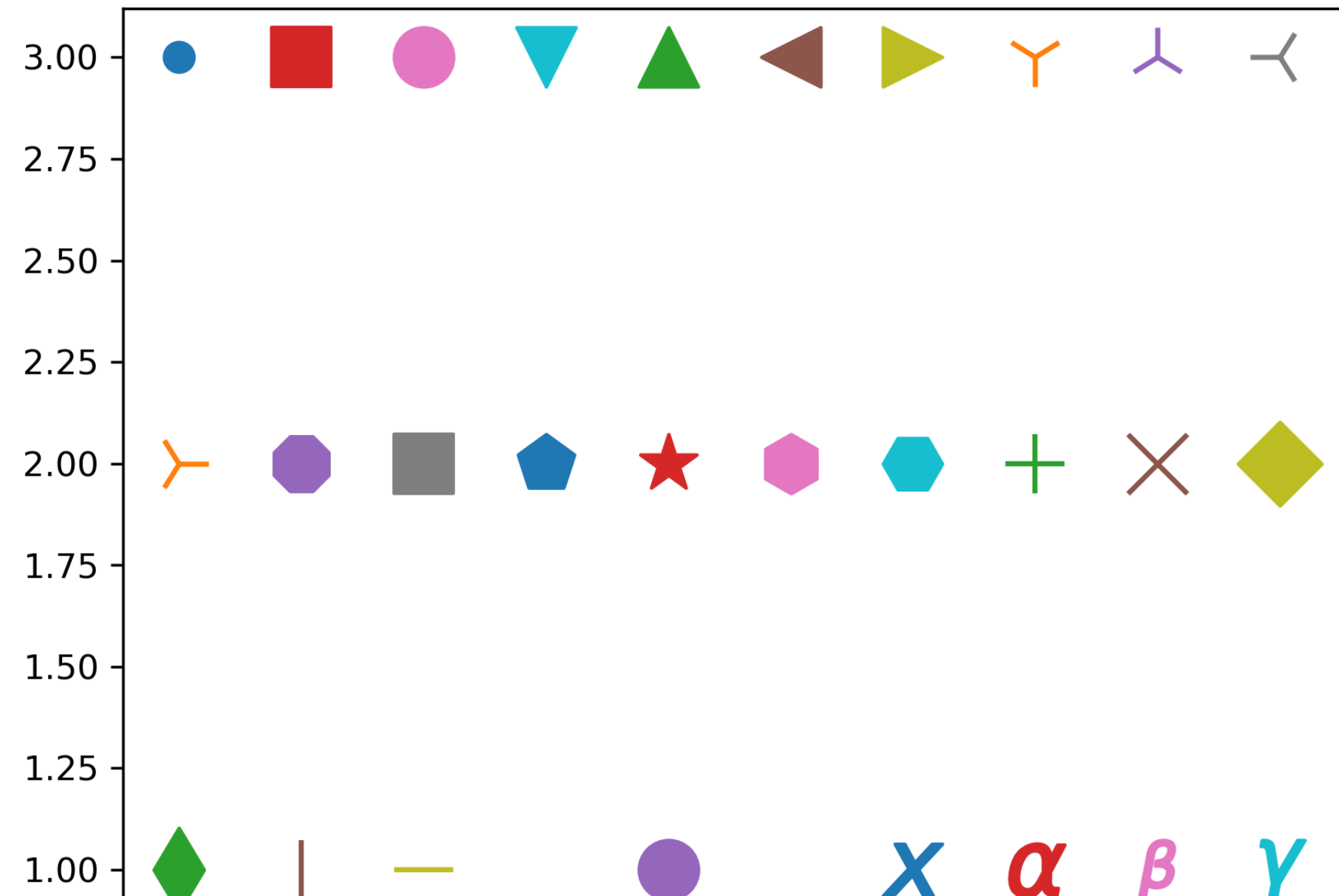
```
for i in x-1:
```

```
    plt.scatter(x[i], y1[i], s=300, marker=markers1[i])
```

```
    plt.scatter(x[i], y2[i], s=300, marker=markers2[i])
```

```
    plt.scatter(x[i], y3[i], s=300, marker=markers3[i])
```

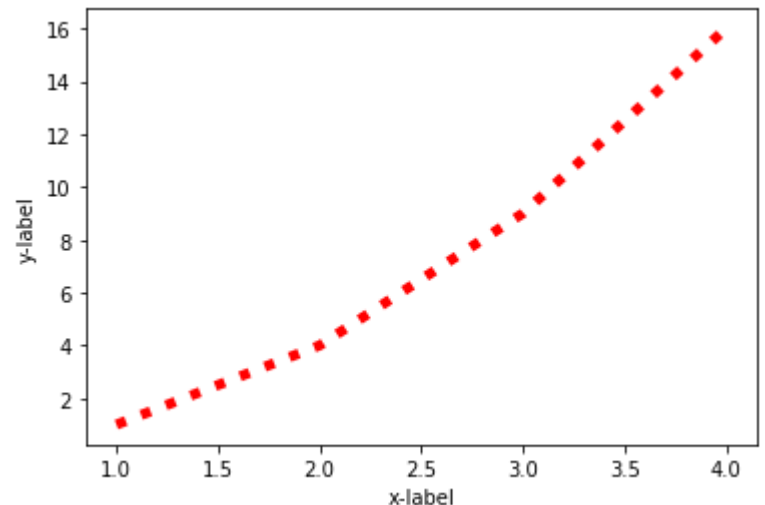
marker.pyの実行結果



ラインを点線にする

```
%matplotlib inline
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4],    # xの値
         [1, 4, 9, 16],   # yの値
         linestyle='dotted', # ラインを点線にする
         linewidth=5,      # ライン幅は5pt
         color='red'       # ラインの色は赤
        )
plt.ylabel('y-label')    # y軸のラベルをプロット
plt.xlabel('x-label')    # x軸のラベルをプロット
plt.show()
```



11,1,5グラフの軸に目盛りを設定する (xticks,yticks)

- xtick(目盛りを挿入する位置,挿入する目盛り)
- ytick(目盛りを挿入する位置,挿入する目盛り)

コード例

positionsとlabelsを設定します

```
positions = [0, np.pi/2, np.pi, np.pi*3/2, np.pi*2]
```

```
labels = ["0° ", "90° ", "180° ", "270° ", "360° "]
```

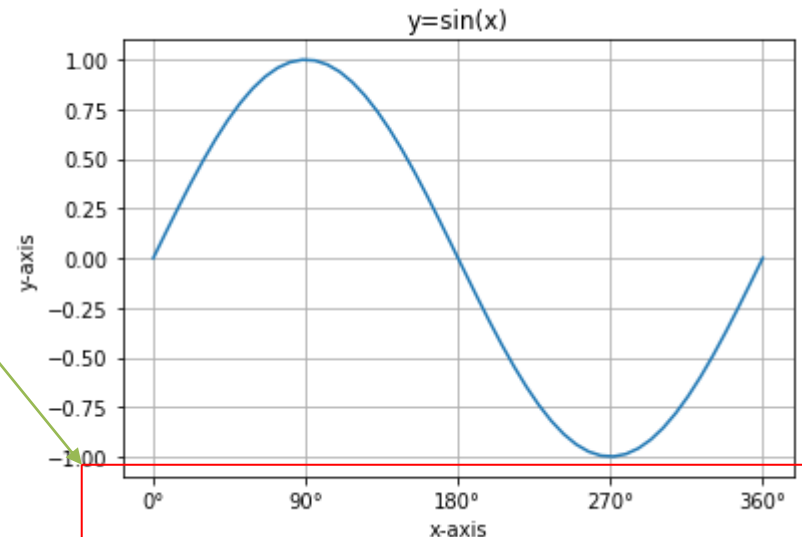
~~グラフのx軸に目盛りを設定してください~~

```
plt.xticks(positions, labels)
```

11, 1, 5コード

```
x = np.linspace(0, 2*np.pi)
y = np.sin(x)
# グラフのタイトルを設定します
plt.title("y=sin(x)")
# グラフのx軸とy軸に名前を設定します
plt.xlabel("x-axis")
plt.ylabel("y-axis")
# グラフにグリッドを表示します
plt.grid(True)
# positionsとlabelsを設定します
positions = [0, np.pi/2, np.pi, np.pi*3/2, np.pi*2]
labels = ["0° ", "90° ", "180° ", "270° ", "360° "]
# グラフのx軸に目盛りを設定してください
plt.xticks(positions, labels)

# データx,yをグラフにプロットし、表示します
plt.plot(x,y)
plt.show()
```

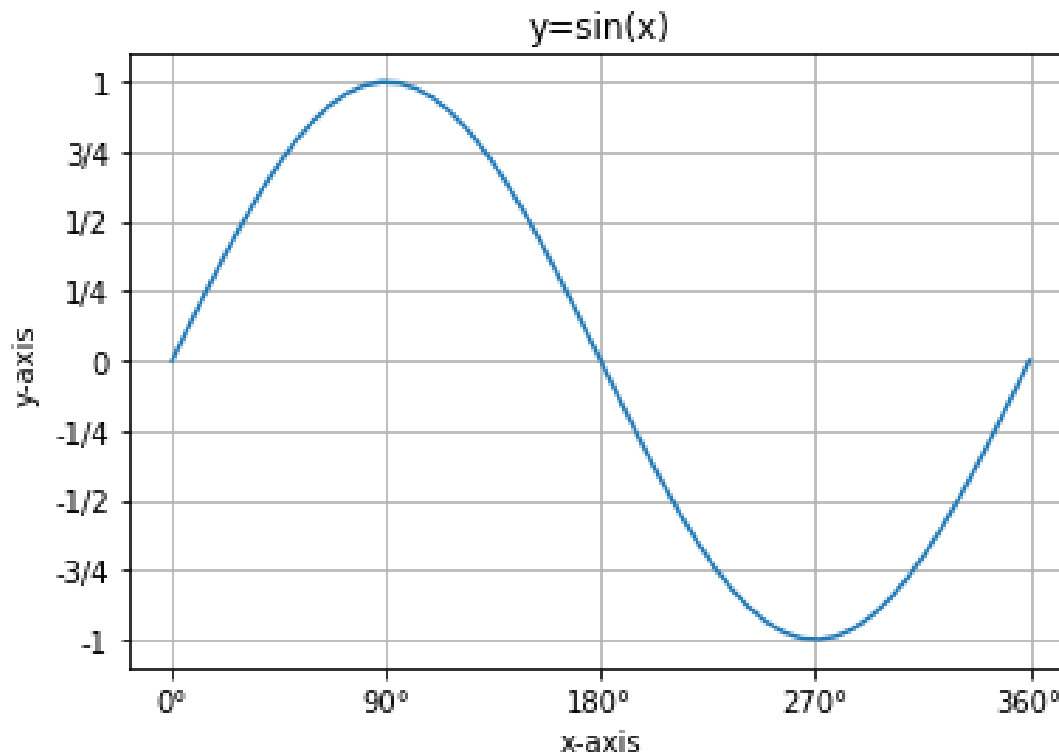


課題

- y_tickを分数表示に変えてみてください

解答

```
positions = [-1, -3/4, -1/2, -1/4, 0, 1/4, 1/2, 3/4, 1]  
labels = ["-1", "-3/4", "-1/2", "-1/4", "0", "1/4", "1/2", "3/4", "1"]  
plt.yticks(positions, labels)
```



演習

(1) 巨人軍の視聴率をグラフにしてください

(rate.csv)

(2) SQLから読み込みグラフにしてください

(graf1.py)

複数のデータを可視化する

1つのグラフに2種類のデータをプロットする

11,1,2 plotを二度書くと重なったグラフが描ける

```
plt.title("graphs of trigonometric functions")
# グラフのx軸とy軸に名前を設定します
plt.xlabel("x-axis")
plt.ylabel("y-axis")
# グラフにグリッドを表示します
plt.grid(True)
# グラフのx軸にラベルを設定します
plt.xticks(positions, labels)
# データx, y1をグラフにプロットし、黒で表示してください
plt.plot(x, y1, color="k")
# データx, y2をグラフにプロットし、青で表示してください
plt.plot(x, y2, color="b")
plt.show()
```

上3つだけの部分を書く

Figureオブジェクトを作成します

```
fig = plt.figure(figsize=(9, 6))
```

```
ax = fig.add_subplot(2, 3, 1)
```

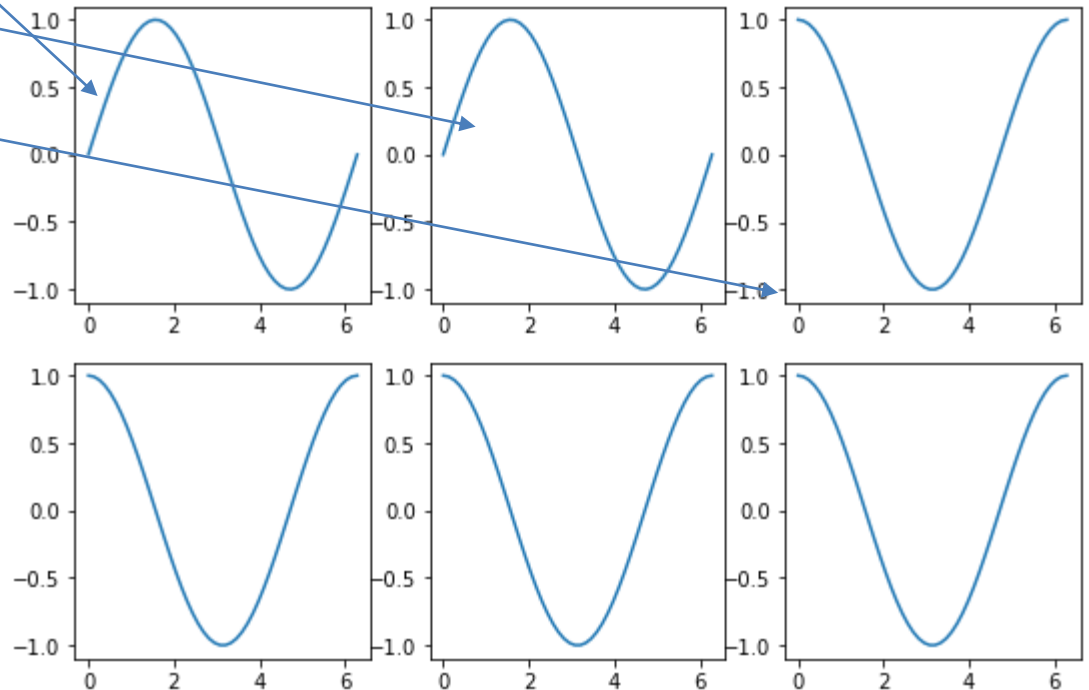
```
ax.plot(x,y)
```

```
ax = fig.add_subplot(2, 3, 2)
```

```
ax.plot(x,y)
```

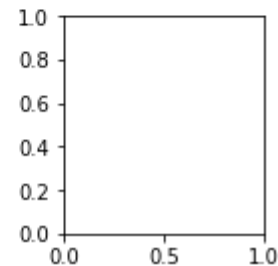
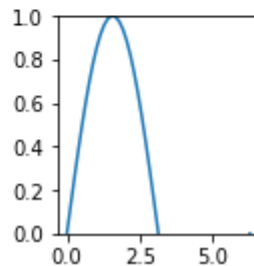
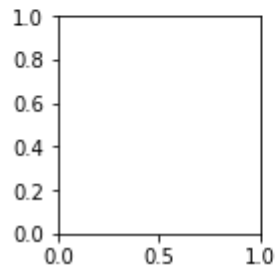
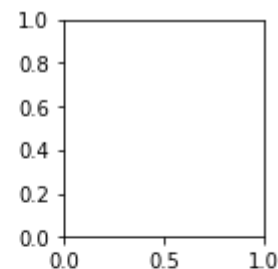
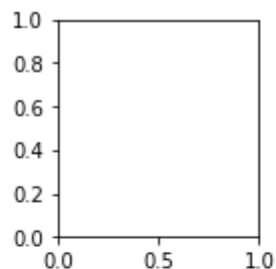
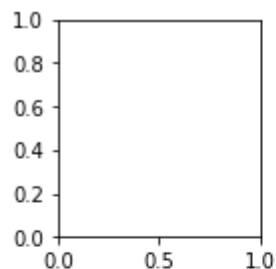
```
ax = fig.add_subplot(2, 3, 3)
```

```
ax.plot(x,y)
```

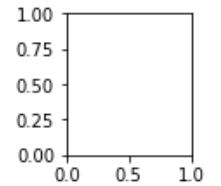
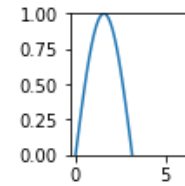
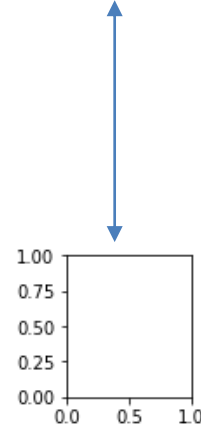
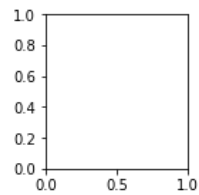
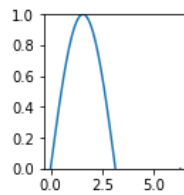
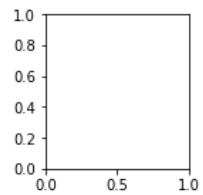
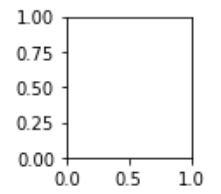
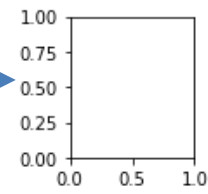
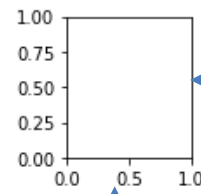
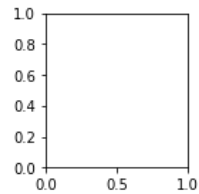
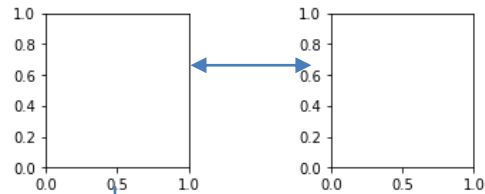


`plt.subplots_adjust(wspace=1, hspace=1)`

- `subplots_adjust(wspace=横間隔をあける割合, 縦間隔をあける割合)`



$(wspace=1, hspace=1) \nless (wspace=2, hspace=2)$



11.3.4サブプロット内のグラフの表示範囲を設定する

set_xlim,set_ylim,set_xlabel,set_ylabel,set_title

set_xlim (範囲)

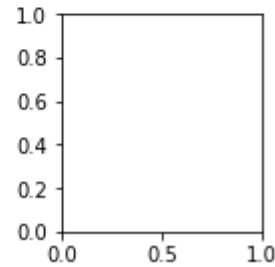
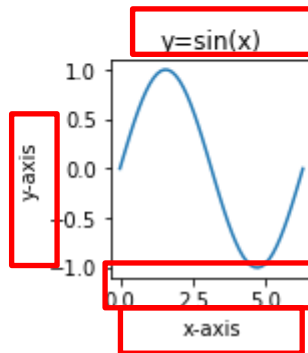
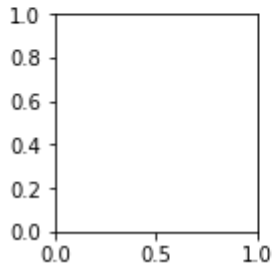
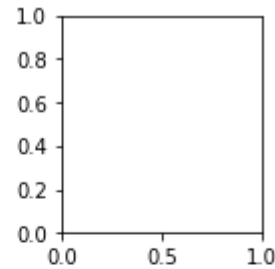
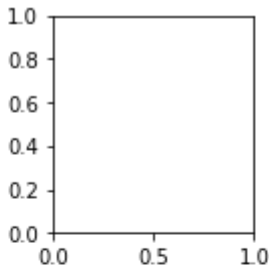
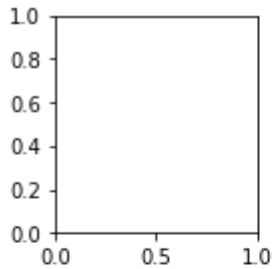
set_ylim (範囲)

set_title(“タイトル”)

set_xlabel(“x軸の名前”)

set_ylabel(“y軸の名前”)

set_xlim,set_xlabel,set_ylabel,set_title



11.3.6サブプロット内のグラフにグリッド を表示する

```
ax.grid(True)
```

11.3.7サブプロット内のグラフに目盛りを設定する

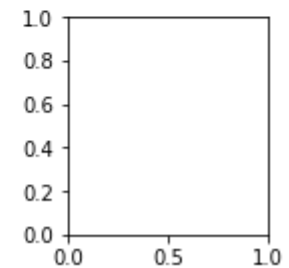
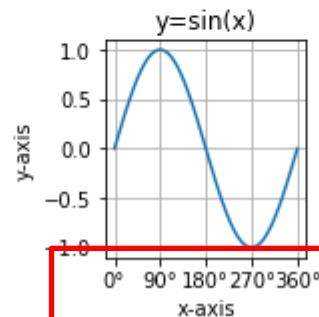
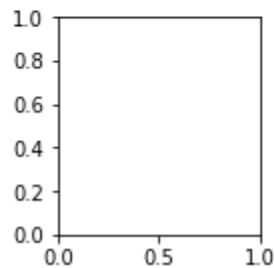
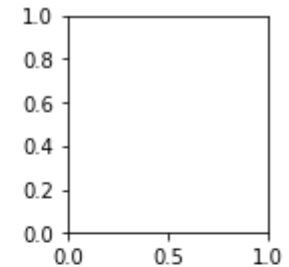
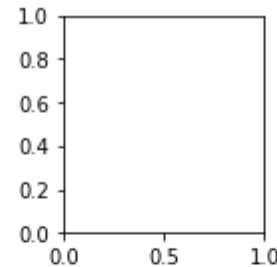
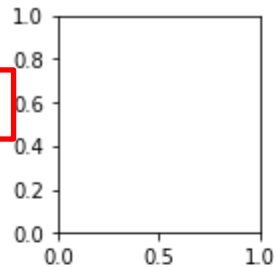
- `set_xtick(“挿入位置リスト”)`
- `set_xticklabels(“目盛りのリスト”)`

11.3.7

```
positions = [0, np.pi/2, np.pi, np.pi*3/2, np.pi*2]  
labels = ["0° ", "90° ", "180° ", "270° ", "360° "]
```

`ax.set_xticks(position)`

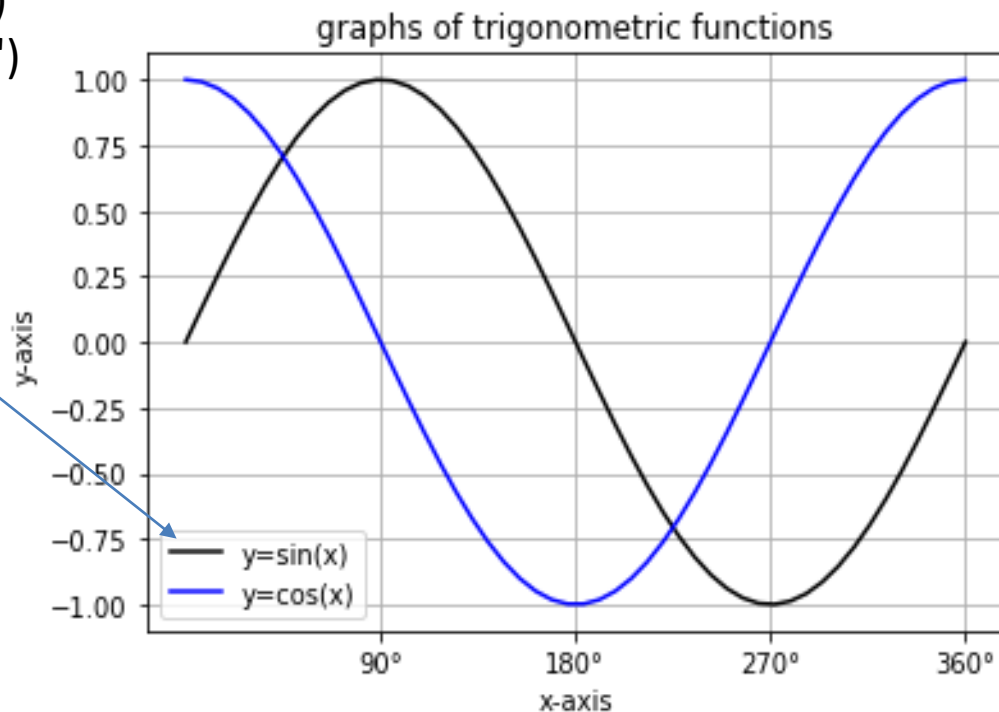
`ax.set_xticklabels(labels)`



11,2系列ラベルを設定する(legend)

```
plt.title("graphs of trigonometric functions")  
plt.xlabel("x-axis")  
plt.ylabel("y-axis")  
plt.grid(True)  
plt.xticks(positions, labels)  
plt.plot(x, y1, color="k", label="y=sin(x)")  
plt.plot(x, y2, color="b", label="y=cos(x)")  
plt.legend(["y=sin(x)", "y=cos(x)"])
```

リストで区切ると複数の凡例が入る



locパラメータ

```
plt.legend(["y=sin(x)", "y=cos(x)"], loc='upper right')
```

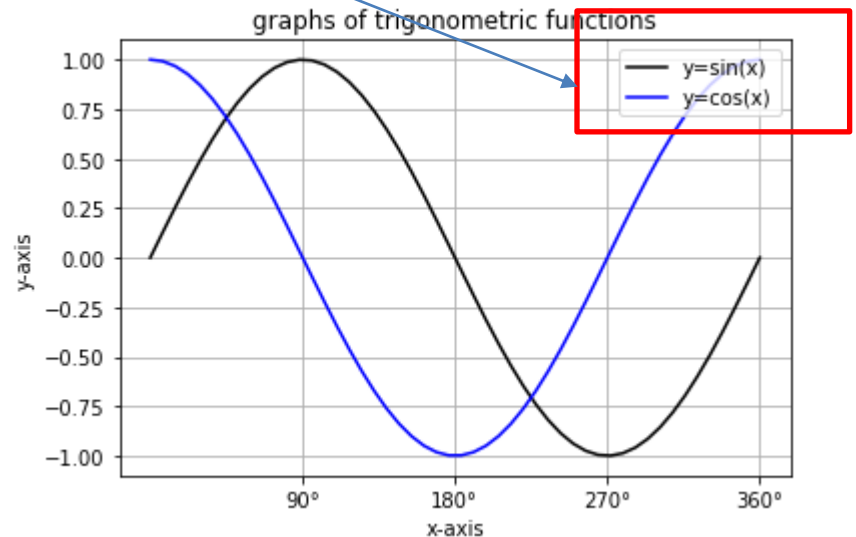
locパラメータは凡例を

loc= 'upper right' 上右方向

loc= 'upper left' 上左方向

loc= 'lower right' 下右方向

loc= 'lower left' 下左方向

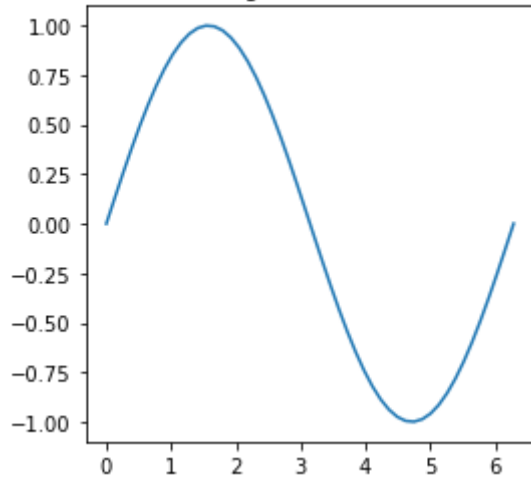


11,3,1図の大きさを設定する figure

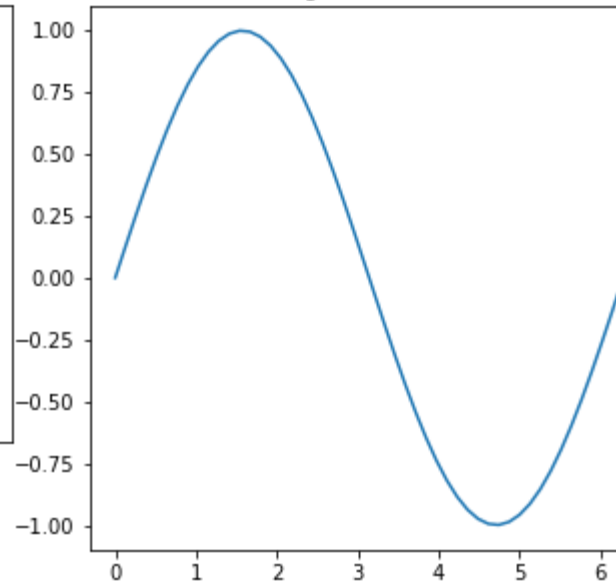
- `figure(figsize(横の大きさ,縦の大きさ))`
(単位はインチ)

figsize=(x,y)

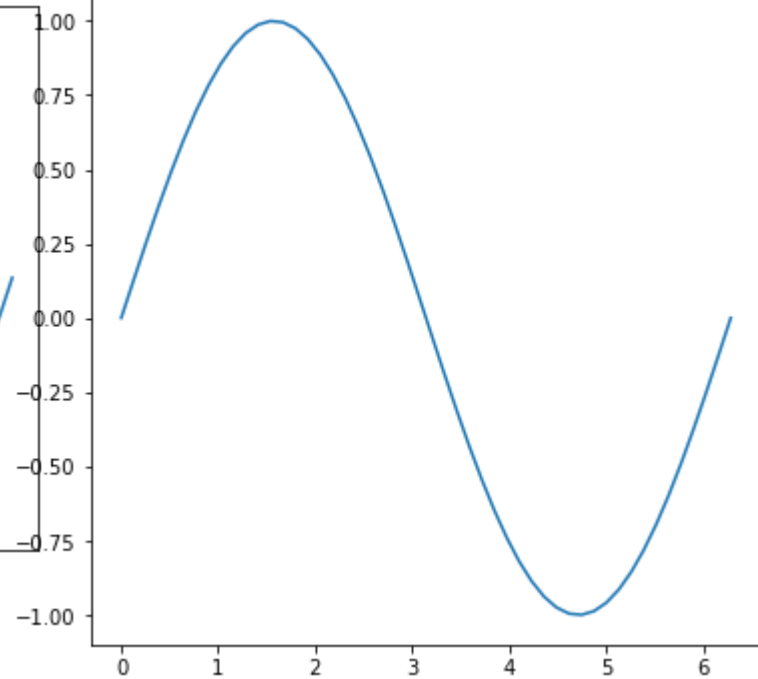
figsize=(4, 4)



figsize=(5, 5)



figsize=(6, 6)



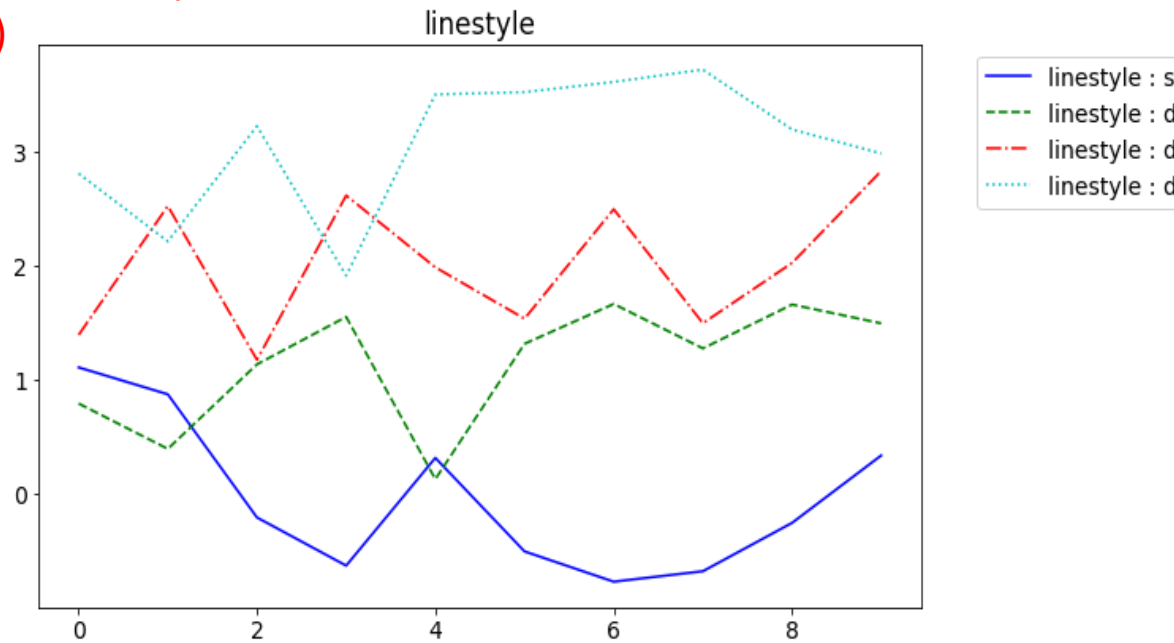
11,3,2サブプロットを作成する (figureオブジェクトとadd_subplot)

```
fig = plt.figure(figsize=(9, 6))
ax = fig.add_subplot(2, 3, 5)
ax.plot(x,y)
axi = []
for i in range(6):
    if i==4:
        continue
    fig.add_subplot(2, 3, i+1)
plt.show()
```

```

import matplotlib.pyplot as plt
import numpy as np
x = np.arange(10)
y0 = np.random.normal(0, 0.5, 10)
y1 = np.random.normal(1, 0.5, 10)
y2 = np.random.normal(2, 0.5, 10)
y3 = np.random.normal(3, 0.5, 10)
plt.rcParams["font.size"] = 14
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(1, 1, 1, title="linestyle")
ax.plot(x, y0, linestyle="-", c="b", label="linestyle : solid, '-'")
ax.plot(x, y1, linestyle="--", c="b", label="linestyle : dashed, '--'")
ax.plot(x, y2, linestyle="-.", c="b", label="linestyle : dashdot, '-.'"")
ax.plot(x, y3, linestyle=":", c="b", label="linestyle : dotted, ':'")
ax.legend(bbox_to_anchor=(1.05, 1))
plt.show()

```



課題

temperature.csvに各地域の気温データがあります。これを折れ線グラフにしてください
(先頭から20ぐらいで)

(1)一つのグラフに複数書く場合

(2)複数のグラフに書く場合(6つのグラフ)時間のある人はしてください

2次元のグラフを書く

$y = x^2 + 2x - 1$ のグラフを書く

```
def f(x):  
    return x*x+2*x-1  
  
x=np.linspace(-3,3,100)  
print(np.round(x,2))  
  
plt.plot(x,f(x))  
plt.grid(True)  
plt.show()
```

課題

$y = x^3 + 2x - 1$ のグラフを書く

```
# -*- coding: utf-8 -*-  
from sympy import *
```

```
var("a:z")      # a～zまで変数として扱う
```

```
#f = x**2 + 3*x + 2    # 関数f(x)の定義
```

```
f = x*x+2*x-1
```

```
y=[]
```

```
x3=[]
```

```
for x2 in np.linspace(-3.0, 3.0, 100):
```

```
    #print(x4)
```

```
    f1 = f.subs([(x, x2)])
```

```
    x3.append(x2)
```

```
    y.append(f1)
```

```
plt.plot(x3,y)
```

```
plt.show()
```

12.3 ヒストグラム

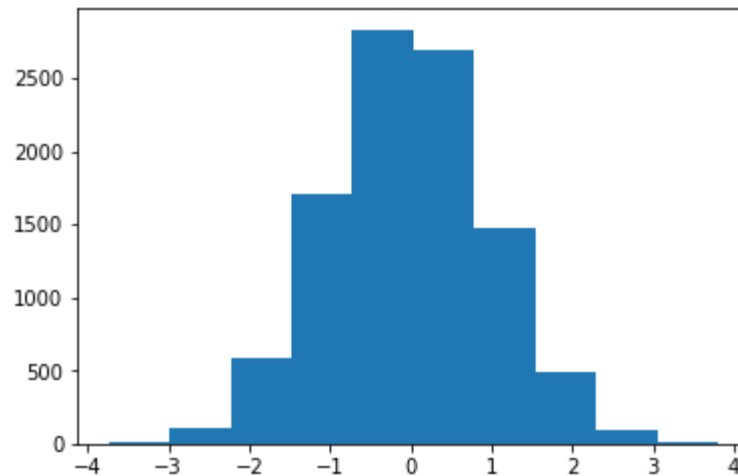
histを使う

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

```
np.random.seed(0)
data = np.random.randn(10000)
```

```
plt.hist(data)
```

```
plt.show()
```



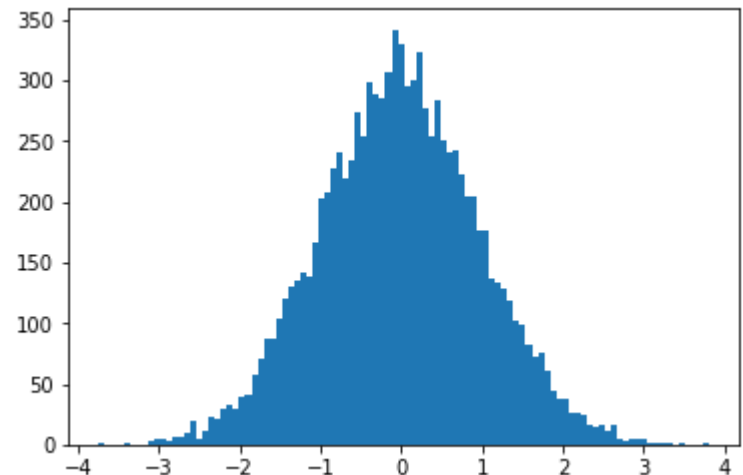
ピン数(階級)を増やす

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

```
np.random.seed(0)  
data = np.random.randn(10000)
```

```
plt.hist(data, bins=100)
```

```
plt.show()
```



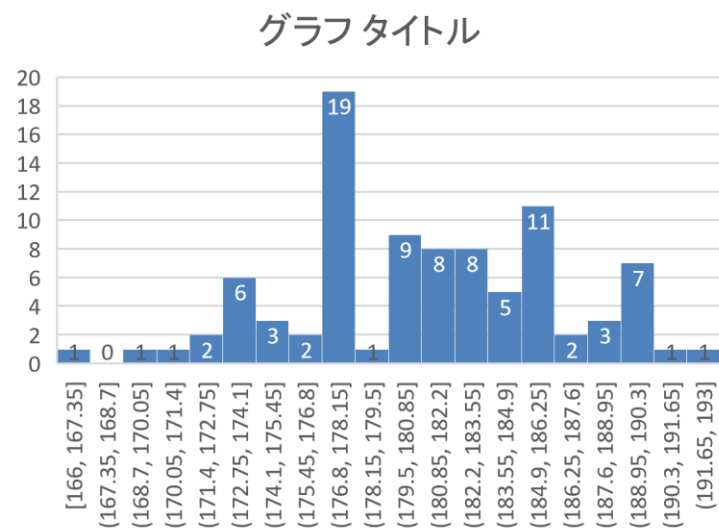
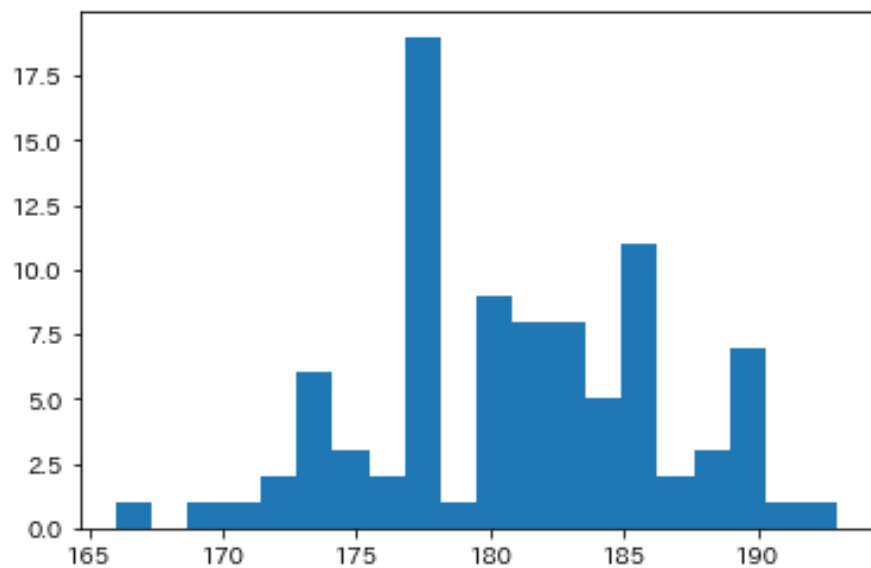
問題

- 巨人軍のデータから身長,体重のヒストグラムをつくりなさい

解答例

```
#-----  
#巨人軍の身長ヒストグラム  
#-----  
  
import pandas as pd  
import matplotlib.pyplot as plt  
%matplotlib inline  
df = pd.read_csv('g.csv')  
shisyo = np.array  
shisyo = df.values  
height=[]  
for s in shisyo:  
    #print(s[3])  
    height.append(s[3])  
plt.hist(height, bins=20)  
  
plt.show()
```

表示例



12.5 円グラフ

12.5.1 円グラフ

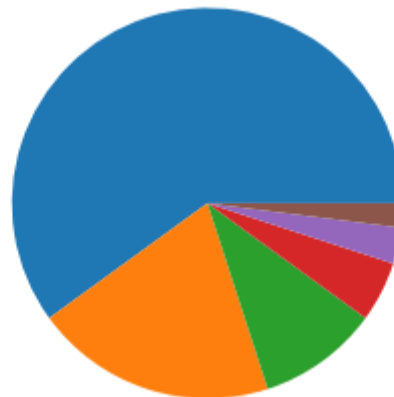
```
import matplotlib.pyplot as plt  
%matplotlib inline  
data = [60, 20, 10, 5, 3, 2]
```

```
plt.pie(data)
```

```
# 円グラフを円楕円から真円にしてください
```

```
plt.axis("equal")
```

```
plt.show()
```

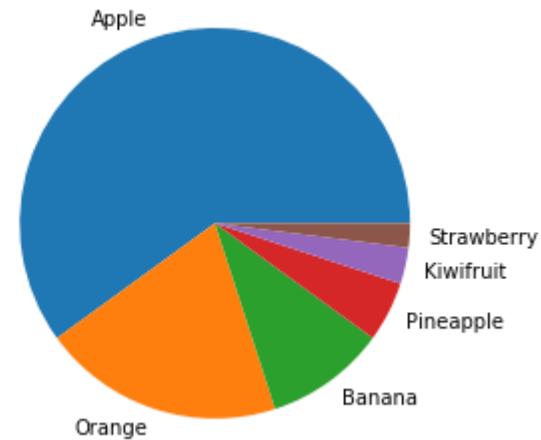


円グラフにラベルを設定する

```
data = [60, 20, 10, 5, 3, 2]
```

```
labels = ["Apple", "Orange", "Banana", "Pineapple", "Kiwifruit", "Strawberry"]
```

```
plt.pie(data, labels=labels)
```



12.5.3特定の要素を目立たせる

```
data = [60, 20, 10, 5, 3, 2]
```

```
labels = ["Apple", "Orange", "Banana", "Pineapple", "Kiwifruit", "Strawberry"]
```

```
explode = [0, 0, 0.1, 0, 0, 0]
```

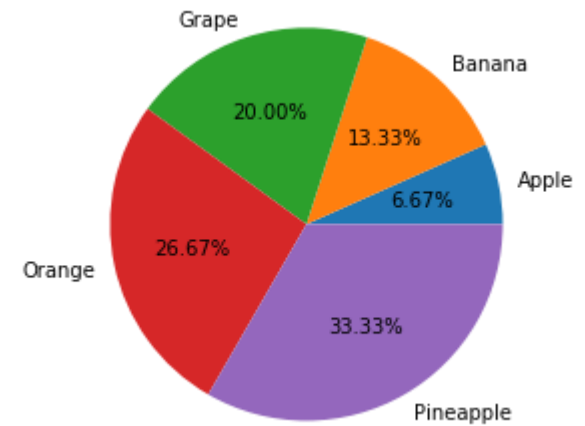
```
plt.pie(data, labels=labels, explode=explode)
```


円グラフ

```
%matplotlib inline
import matplotlib.pyplot as plt
```

```
values = [100, 200, 300, 400, 500] # グラフ要素の値
labels = [                                # グラフ要素のラベル
    'Apple', 'Banana', 'Grape', 'Orange', 'Pineapple'
]
```

```
plt.pie(x=values,                # グラフ要素の値を設定
        labels=labels,          # グラフ要素のラベルを設定
        autopct='%.2f%%')      # 構成割合として小数点以下2桁までをプロット
plt.axis('equal')               # グラフを
plt.show()
```



時計回りから表示

```
plt.pie(x=values,  
        labels=labels,  
        autopct='%.2f%%',  
        startangle=90,  
        counterclock=False  
    )
```

```
plt.axis('equal')
```

```
plt.show()
```

グラフ要素の値を設定

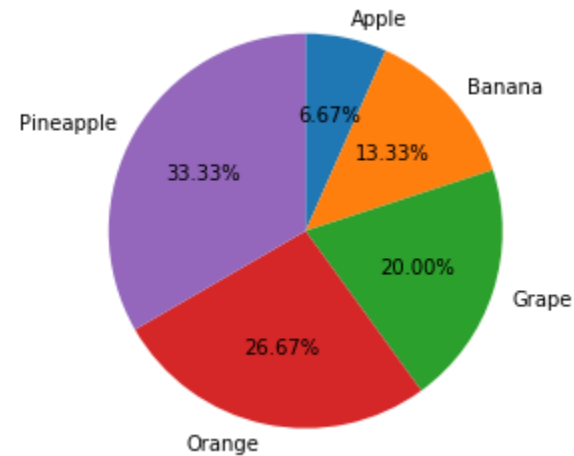
グラフ要素のラベルを設定

構成割合として小数点以下2桁までをプロット

90度(真上)の位置から開始

時計回りにする

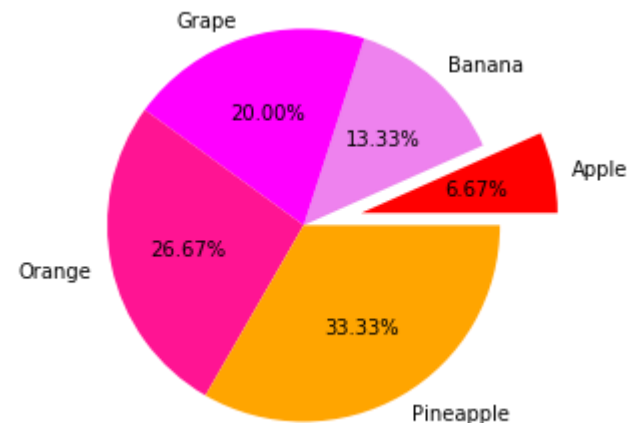
グラフを真円にする



```
# 要素のカラーを指定するリスト エッジライン
setcolors = ['red', 'violet', 'fuchsia', 'deeppink', 'orange']
plt.pie(x=values,          # グラフ要素の値を設定
        labels=labels,     # グラフ要素のラベルを設定
        colors=setcolors,  # グラフ要素のカラーを設定
        wedgeprops={
            'linewidth': 3,  # エッジラインの幅は3
            'edgecolor': 'white' # エッジラインの色はホワイト
        },
        labeldistance=0.5,  # ラベルを円周内の50%の位置に表示
        textprops={
            'color': 'white', # ラベルテキストのカラーはホワイト
            'weight': 'bold'} # 太字にする
    )
plt.axis('equal')          # グラフを真円にする
plt.show()
```



```
plt.pie(x=values,          # グラフ要素の値を設定
        labels=labels,     # グラフ要素のラベルを設定
        autopct='%0.2f%%', # 構成割合として小数点以下2桁までをプロット
        colors=setcolors,  # グラフ要素のカラーを設定
        explode=[0.3, 0, 0, 0, 0] # 1番目の要素の中心位置を円周上から0.3にする
    )
plt.axis('equal')          # グラフを真円する
plt.show()
```



問題

1, TestDb.dbのテーブルplayerの

(1) 体重70kg未満と

(2) 70kgから79kg

(3) 80kgから89kg

(4) 90kgから99kg

(5) 100kg以上の人数を求めて円グラフを書いて
みてください。SQLで検索してください

SQL(engraf1.py)

select count(*) from player where 体重<70; 22人

select count(*) from player where 体重>=70 and
体重<=79; 206人

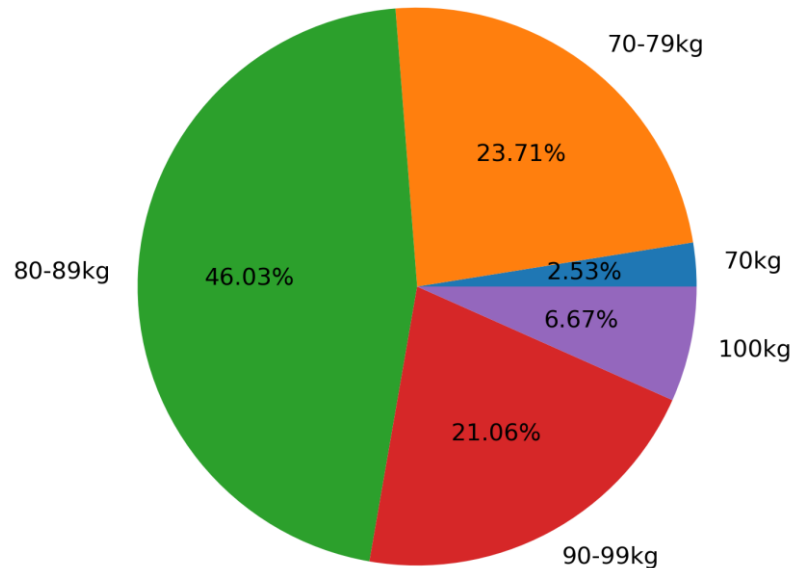
select count(*) from player where 体重>=80 and
体重<=89; 400人

select count(*) from player where 体重>=90 and
体重<=99; 183人

select count(*) from player where 体重>=100
; 58人

課題(コードなし)

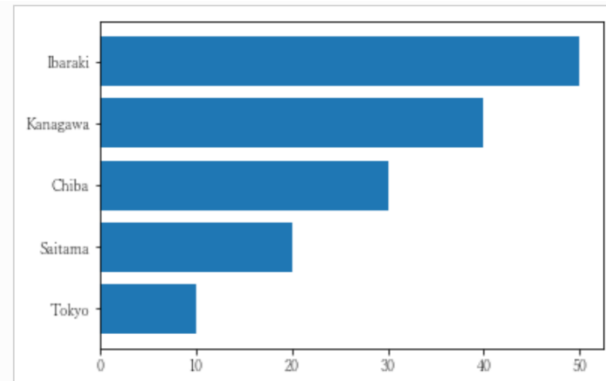
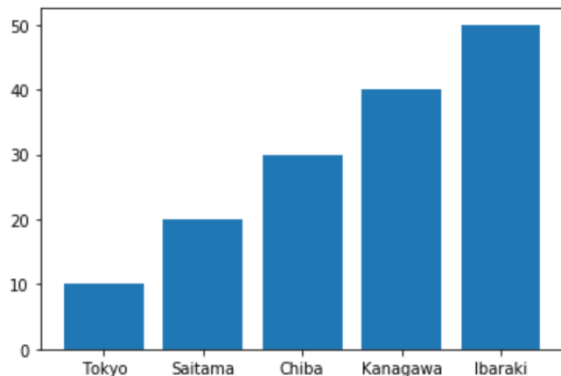
先ほどのSQLをコードに埋め込んで
人数を自動的に算出してグラフを書く方法を
考えてみてください。



12.2棒グラフ

棒グラフ

```
import matplotlib.pyplot as plt  
plt.bar( ['Tokyo', 'Saitama', 'Chiba', 'Kanagawa',  
         'Ibaraki'], [10, 20, 30, 40, 50] )  
# barh()にすると棒グラフが横になる  
plt.show()
```



複数の棒グラフ

```
xx1 = [1, 2, 3]
```

```
yy1 = [4, 5, 6]
```

```
xx2 = [1.3, 2.3, 3.3]
```

```
yy2 = [2, 4, 1]
```

```
label_x = ['Result1', 'Result2', 'Result3']
```

```
# 1つ目の棒グラフ
```

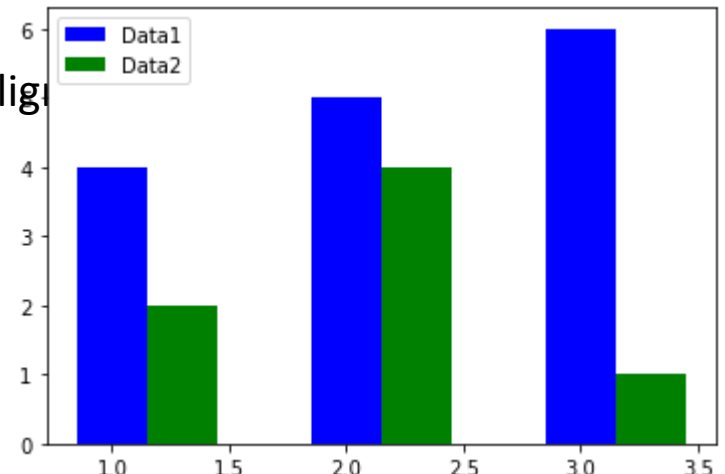
```
plt.bar(xx1, yy1, color='b', width=0.3, label='Data1', align="center")
```

```
# 2つ目の棒グラフ
```

```
plt.bar(xx2, yy2, color='g', width=0.3, label='Data2', align="center")
```

```
# 凡例
```

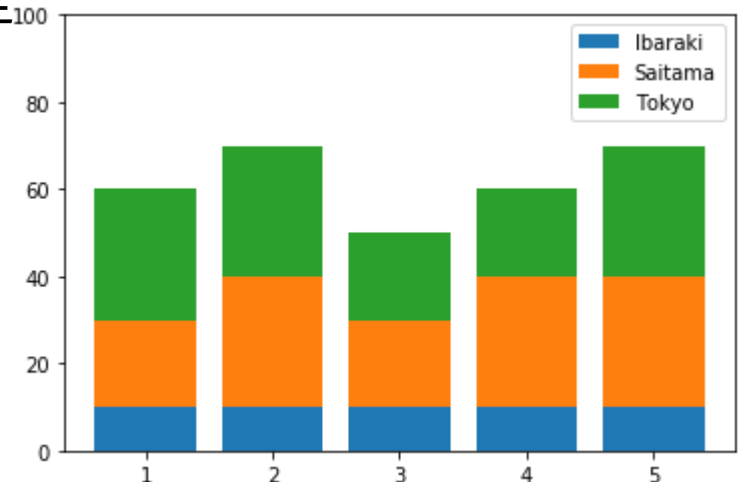
```
plt.legend(loc=2)
```



複数の棒グラフ

```
import numpy as np
import matplotlib.pyplot as plt
x_list = [0, 1, 2, 3, 4] # 目盛りの値の設定
y_list = np.arange(0, 101, 20)
tokyo = [30, 30, 20, 20, 30] # グラフの値
saitama = [20, 30, 20, 30, 30]
ibaraki = [10, 10, 10, 10, 10]
tokyo_bottom = np.array(saitama) + np.array(ibaraki) # 一番に積み上がる棒グラフの
    かさ上げ用
plt.bar(x_list, ibaraki, label = 'Ibaraki') # グラフデータの設定
plt.bar(x_list, saitama, label = 'Saitama', bottom = ibaraki)
plt.bar(x_list, tokyo, label = 'Tokyo' , bottom = tokyo_bottom)
plt.xticks(x_list, (['1', '2', '3', '4', '5'])) # 目盛りの設定
plt.yticks(y_list)
plt.legend() # 凡例の表示
plt.show()# グラフの表示
```

Tokyo
Saitama
Ibaraki



- 下からtokyo_bottom , saitama, ibaraki

tokyo = [30, 30, 20, 20, 30]

saitama = [20, 30, 20, 30, 30]

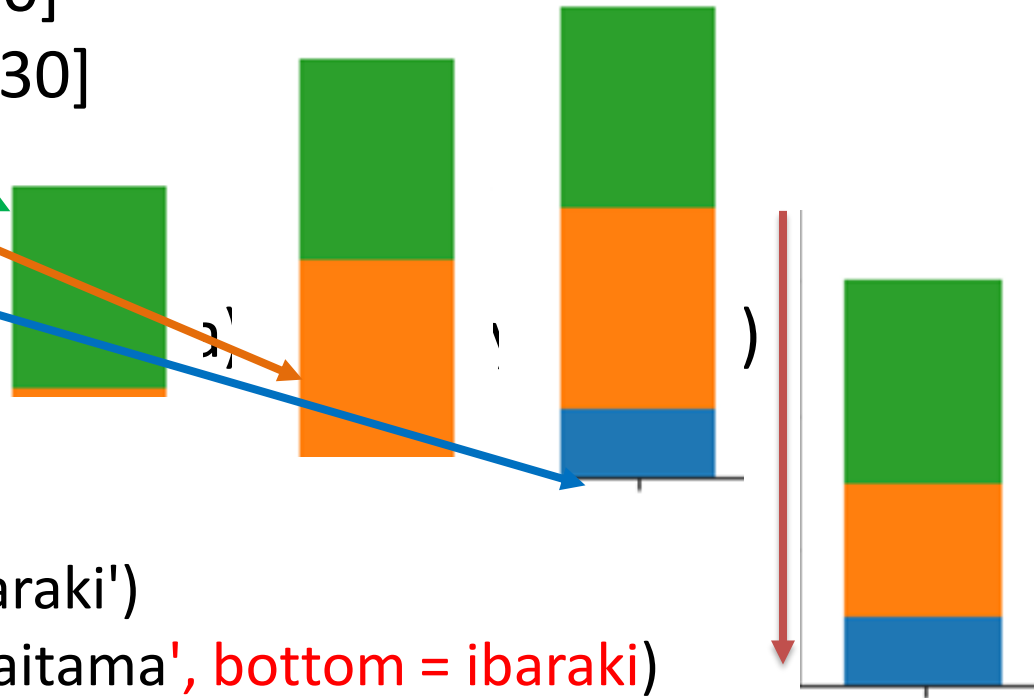
ibaraki = [10, 10, 10, 10, 10]

- tokyo_bottom = np.array

```
plt.bar(x_list, ibaraki, label = 'Ibaraki')
```

```
plt.bar(x_list, saitama, label = 'Saitama', bottom = ibaraki)
```

```
plt.bar(x_list, tokyo, label = 'Tokyo' , bottom = tokyo_bottom)
```



Pandasを使った場合

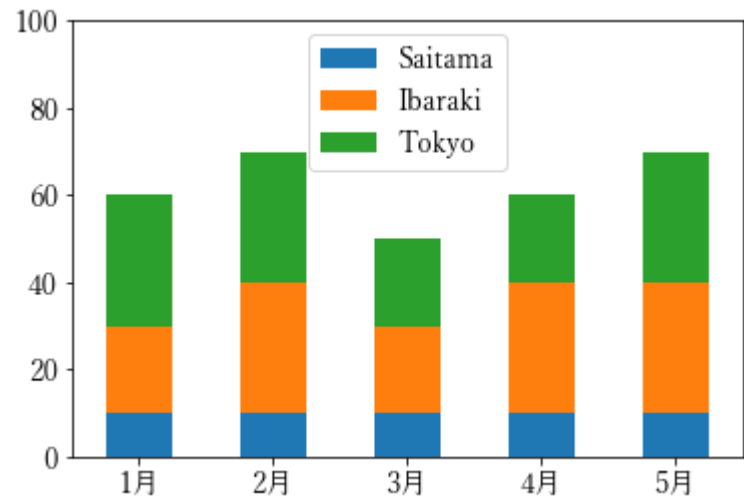
```
import pandas as pd
import matplotlib.pyplot as plt

plt.rcParams["font.family"] = 'Yu Mincho'

df = pd.DataFrame(
    [
        [10, 20, 30],
        [10, 30, 30],
        [10, 20, 20],
        [10, 30, 20],
        [10, 30, 30]
    ],
    index = ['1月', '2月', '3月', '4月', '5月'],
    columns = ['Saitama', 'Ibaraki', 'Tokyo']
)

df.plot.bar(stacked = True, yticks = range(0, 120, 20), rot = 0)
#df.plot(kind = 'bar', yticks = range(0, 81, 10))
#df.plot(kind = 'line', yticks = range(0, 51, 10))

plt.show()
```



課題

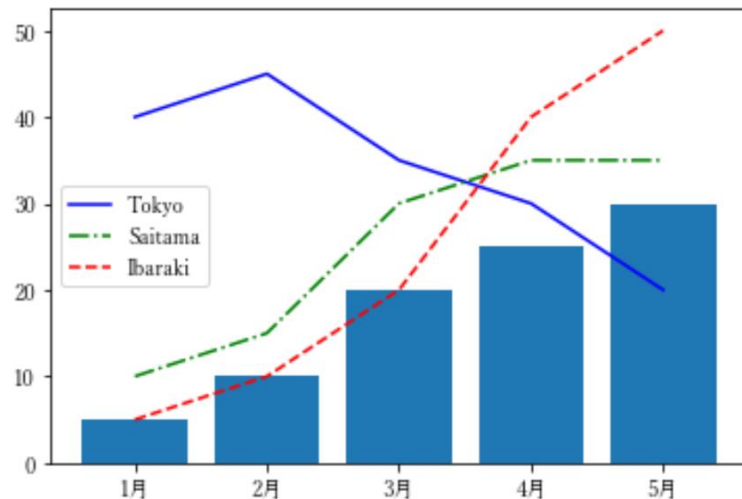
次の表はある中学の実力テスト分布表です。科目別の棒グラフを5つ表示してください。(リストからでもいいです)

	国語	社会	数学	理科	英語
90-100	4	0	8	2	2
80-89	7	17	19	10	13
70-79	18	13	17	6	12
60-69	28	12	23	13	14
50-59	36	17	16	14	26
40-49	27	17	18	20	21
30-39	10	22	17	24	16
0-29	7	39	19	48	33

複数のグラフ

複数のグラフ

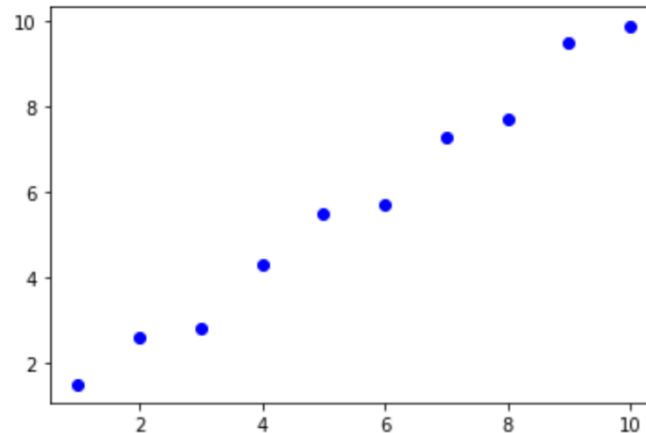
```
import matplotlib.pyplot as plt
x_list = range(0, 5)
plt.plot(x_list, [40, 45, 35, 30, 20], 'b' , label = 'Tokyo') #折れ線グラフ
plt.plot(x_list, [10, 15, 30, 35, 35], '-.g', label = 'Saitama') #棒グラフ
plt.plot(x_list, [ 5, 10, 20, 40, 50], '--r', label = 'Ibaraki') #棒グラフ
plt.bar (x_list, [ 5, 10, 20, 25, 30]) #棒グラフ
plt.xticks(x_list, (['1月', '2月', '3月', '4月', '5月']))
plt.legend()
plt.show()
```



12.4 散布図

散布図

```
import matplotlib.pyplot as plt  
x_list = range(1, 11)  
y_list = [ 1.5, 2.6, 2.8, 4.3, 5.5, 5.7, 7.3, 7.7, 9.5, 9.9]  
plt.scatter(x_list, y_list, c='b')  
#plt.plot( x_list, y_list, 'b*')  
plt.show()
```



plt.scatter

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

np.random.seed(0)
x = np.random.choice(np.arange(100), 100)
y = np.random.choice(np.arange(100), 100)

plt.scatter(x, y)

plt.show()
```

dice = list(range(1, 7))#1から6の値をリスト
print(np.random.choice(dice))#さいころを振る
出力例

```
>>> dice = list(range(1, 7))  
>>> dice  
[1, 2, 3, 4, 5, 6]  
>>> print(np.random.choice(dice))  
6  
>>>
```

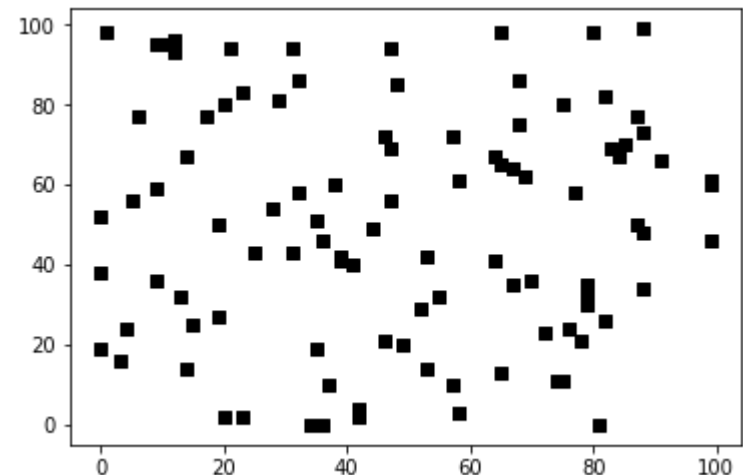
marker="s", color="k"

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

```
np.random.seed(0)
x = np.random.choice(np.arange(100), 100)
y = np.random.choice(np.arange(100), 100)
```

マーカーの種類を四角、色を黒に設定して散布図を作成してください

```
plt.scatter(x, y, marker="s", color="k")
plt.show()
```



plt.scatter(x, y, s=z)

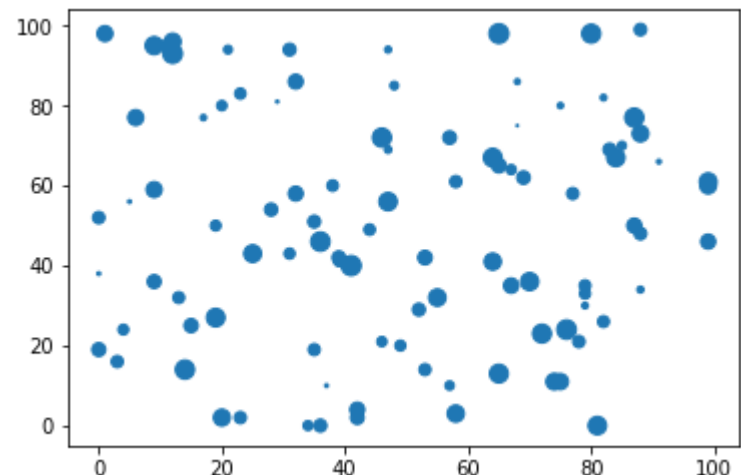
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
np.random.seed(0)
x = np.random.choice(np.arange(100), 100)
y = np.random.choice(np.arange(100), 100)
z = np.random.choice(np.arange(100), 100)
```

zの値に応じて、マーカーの大きさが変わるようにプロットしてください

```
plt.scatter(x, y, s=z)
```

```
plt.show()
```



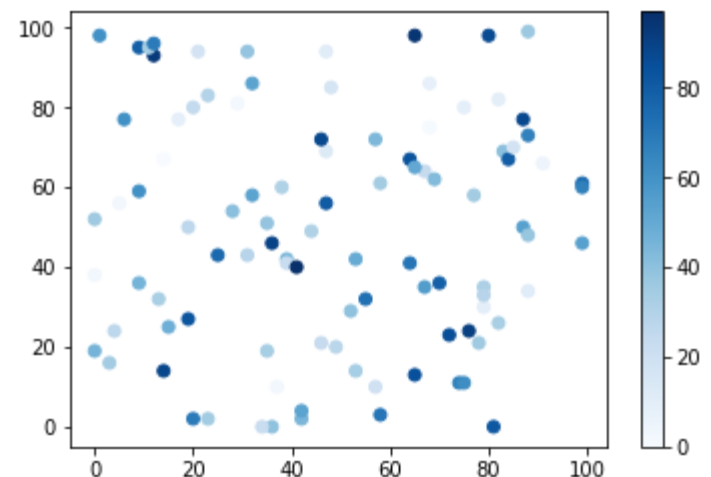
cmap="Blues"とplt.colorbar()

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

```
np.random.seed(0)
x = np.random.choice(np.arange(100), 100)
y = np.random.choice(np.arange(100), 100)
z = np.random.choice(np.arange(100), 100)
```

zの値に応じて、マーカーの濃さが青系統で変わるようにプロットしてください

```
plt.scatter(x, y, c=z, cmap="Blues")
plt.colorbar()
plt.show()
```



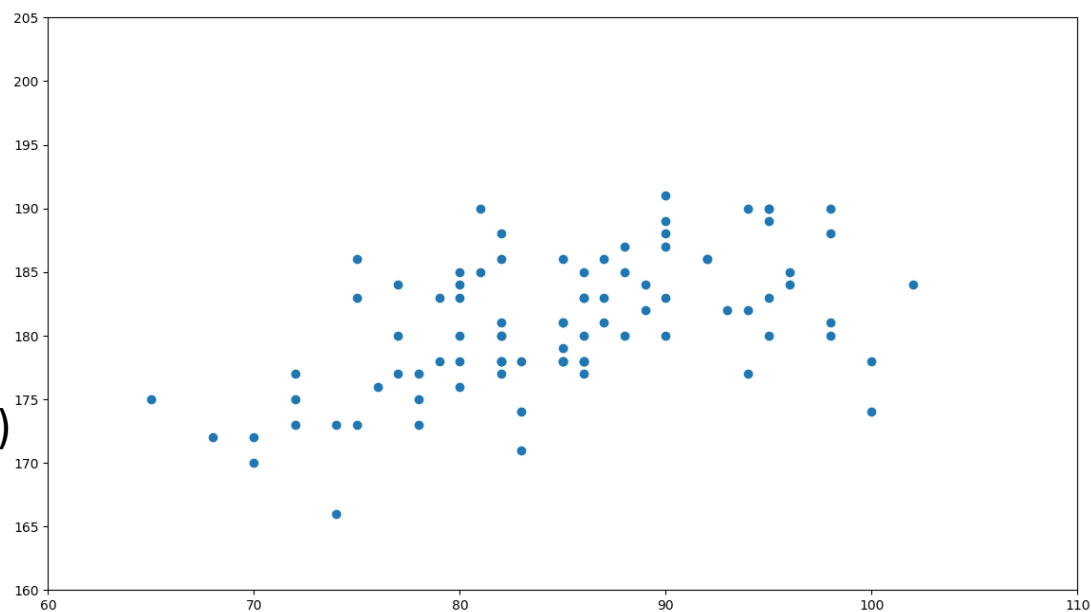
演習

- 巨人軍のデータを使い身長と体重の散布図を作りなさい

解答例

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
df = pd.read_csv('g.csv')#
#df = pd.read_csv('player.csv')
```

```
shisyo = np.array
shisyo = df.values
height=[]
weight=[]
for s in shisyo:
    #print(s[3])#巨人用
    height.append(s[3])
    weight.append(s[4])
plt.figure(figsize=(14, 8), dpi=100)
plt.ylim(160, 205)
plt.xlim(60,110)
plt.scatter(weight, height)
plt.show()
print(shisyo[0])
```



3Dグラフ

3Dグラフ

```
import numpy as np
import matplotlib.pyplot as plt
# 3D描画を行うために必要なライブラリです
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

```
t = np.linspace(-2*np.pi, 2*np.pi)
```

```
X, Y = np.meshgrid(t, t)
```

```
R = np.sqrt(X**2 + Y**2)
```

```
Z = np.sin(R)
```

```
# Figureオブジェクトを作成します
```

```
fig = plt.figure(figsize=(6,6))
```

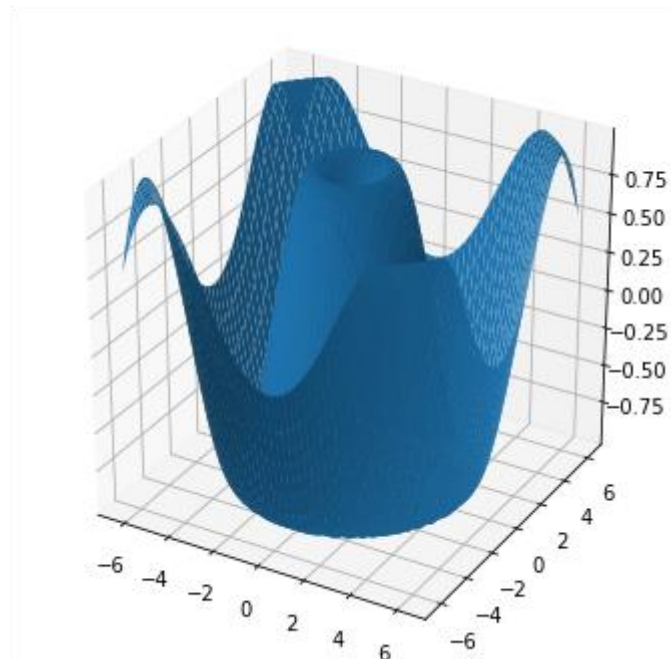
```
# サブプロットaxを作成してください
```

```
ax = fig.add_subplot(1, 1, 1, projection="3d" )
```

```
# プロットして表示します
```

```
ax.plot_surface(X, Y, Z)
```

```
plt.show()
```



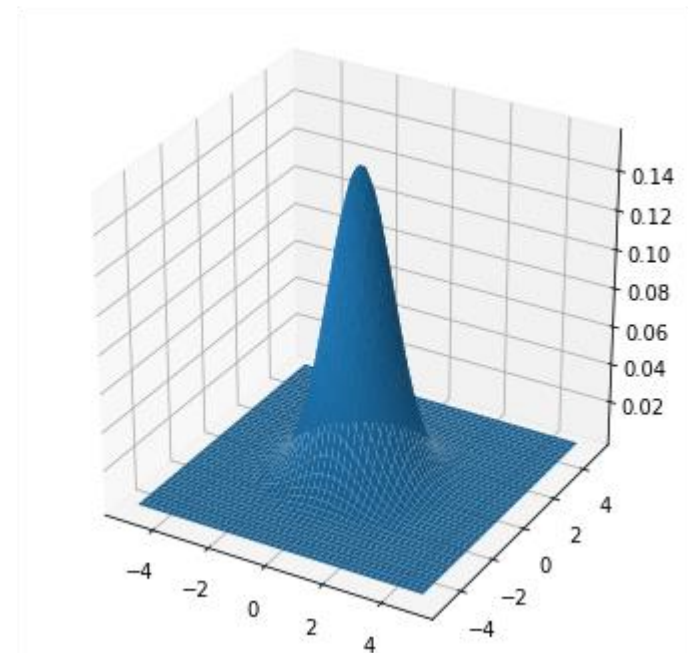
3D2

```
import numpy as np
import matplotlib.pyplot as plt
# 3D描画を行うために必要なライブラリです
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

```
x = y = np.linspace(-5, 5)
X, Y = np.meshgrid(x, y)
Z = np.exp(-(X**2 + Y**2)/2) / (2*np.pi)
```

```
# Figureオブジェクトを作成します
fig = plt.figure(figsize=(6, 6))
# サブプロットaxを作成します
ax = fig.add_subplot(1, 1, 1, projection="3d")
# 曲面を描画して表示してください
ax.plot_surface(X, Y, Z)
```

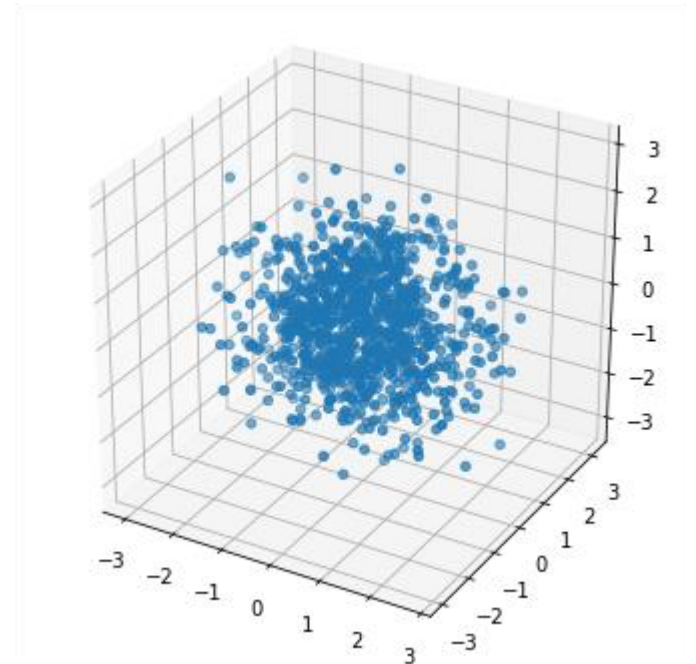
```
plt.show()
```



3D散布図

```
import numpy as np
import matplotlib.pyplot as plt
# 3D描画を行うために必要なライブラリです
from mpl_toolkits.mplot3d import Axes3D
np.random.seed(0)
%matplotlib inline
X = np.random.randn(1000)
Y = np.random.randn(1000)
Z = np.random.randn(1000)
# Figureオブジェクトを作成します
fig = plt.figure(figsize=(6, 6))
# サブプロットaxを作成します
ax = fig.add_subplot(1, 1, 1, projection="3d")
# X,Y,Zを1次元に変換します
x = np.ravel(X)
y = np.ravel(Y)
z = np.ravel(Z)
# 3D散布図を作成してください
ax.scatter3D(x, y, z)

plt.show()
```



問題

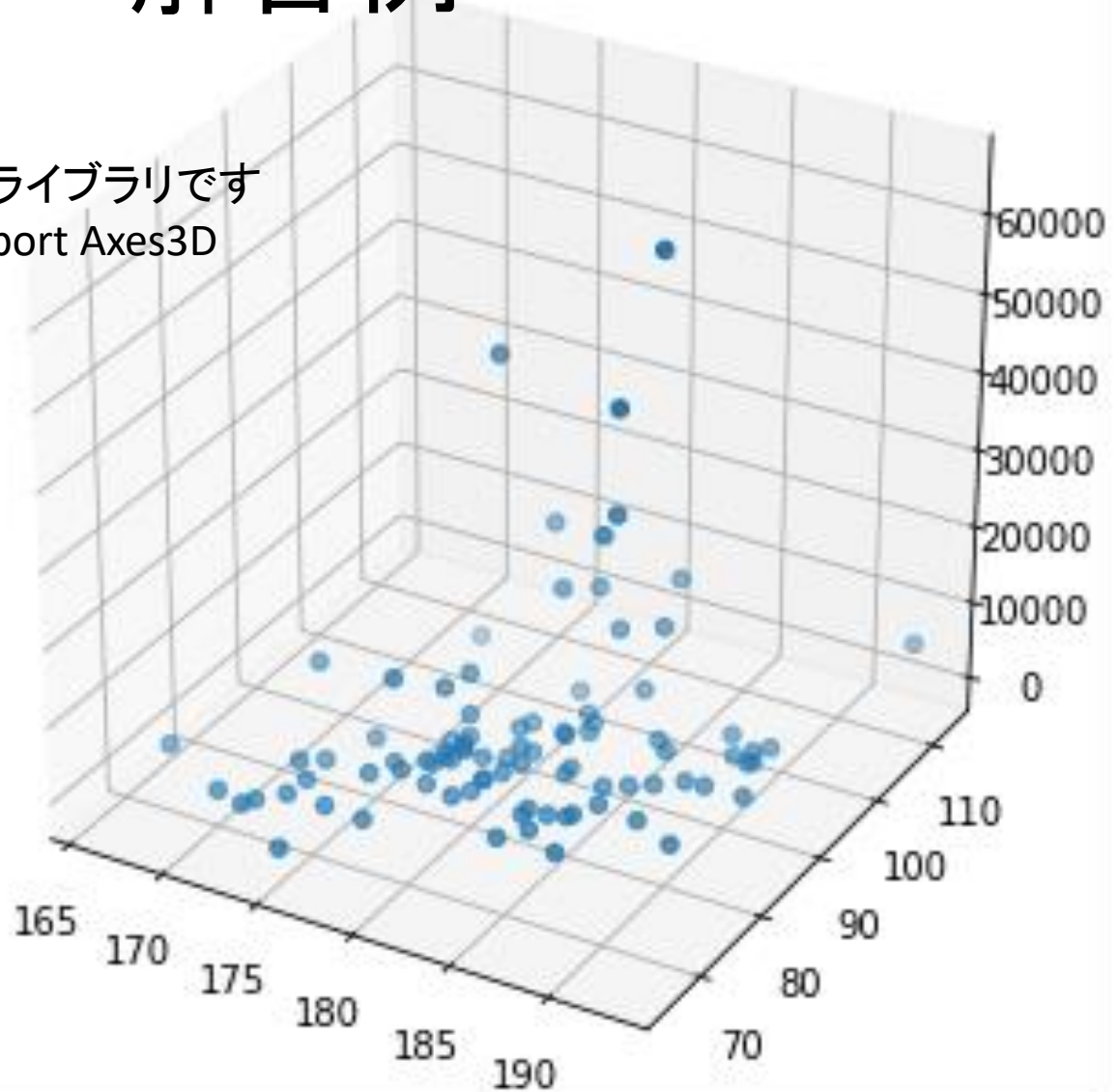
巨人軍のデータから x 身長 y 体重 z 年俸
の三次元散布図を書きなさい

解答例

```
import numpy as np
import matplotlib.pyplot as plt
# 3D描画を行うために必要なライブラリです
from mpl_toolkits.mplot3d import Axes3D
```

```
df = pd.read_csv('g2.csv')#
#df = pd.read_csv('player.csv')
```

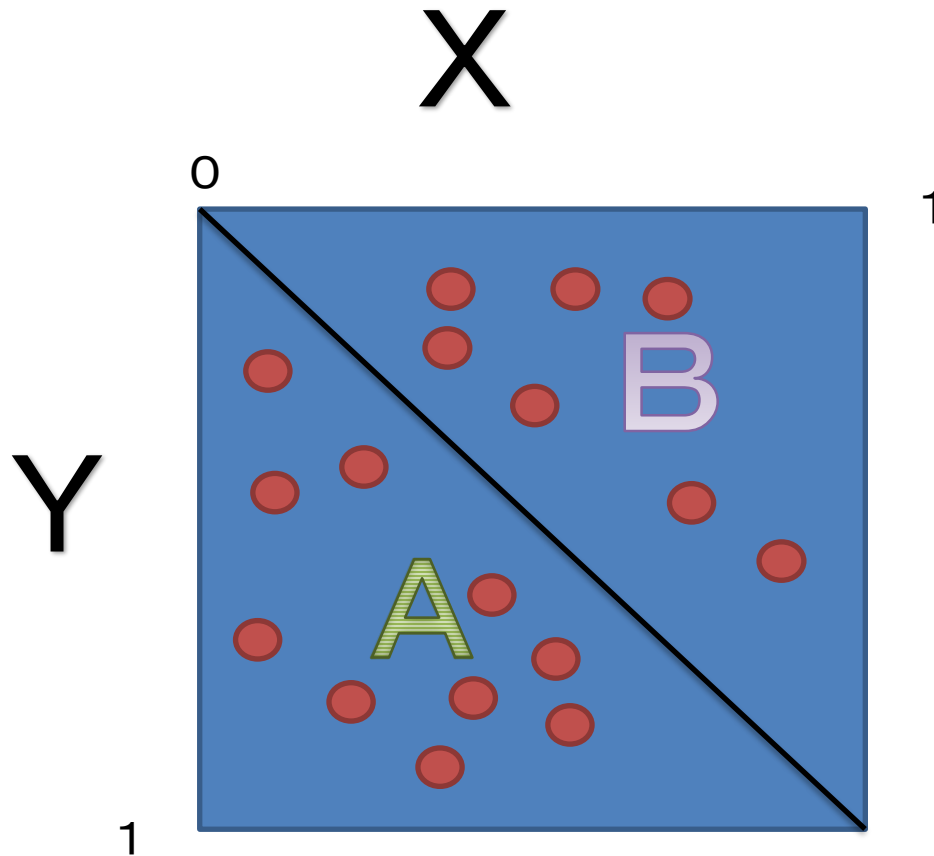
```
shisyo = np.array
shisyo = df.values
height=[]
weight=[]
nenpo=[]
for s in shisyo:
    #print(s[3])#巨人用
    height.append(s[3])
    weight.append(s[4])
    nenpo.append(s[5])
print(height)
print(weight)
print(nenpo)
```



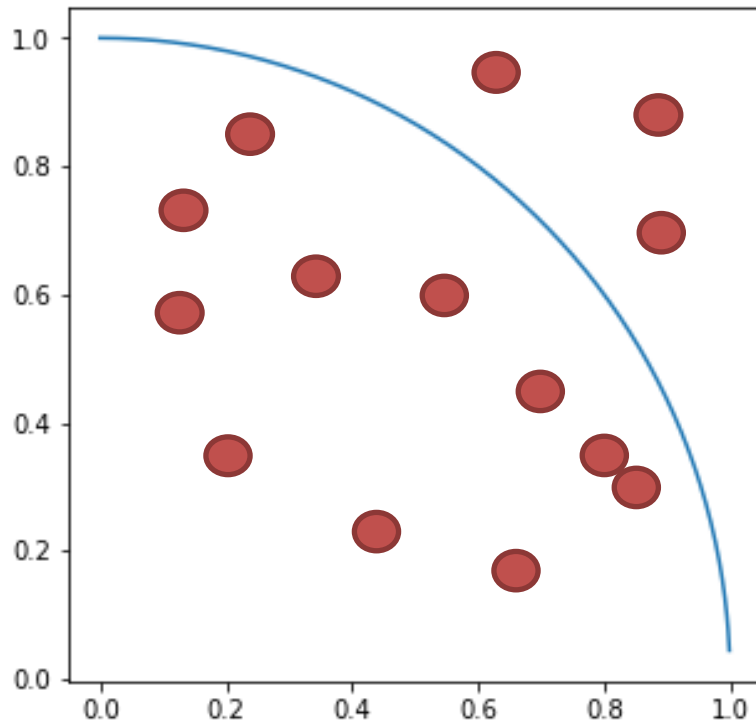
モンテカルロ法

- 乱数より対象の領域に与える場合
対象の領域の面積に比例すると考える

例



例としてXに0から1までの乱数
Yに0から1までの乱数振ったとき
10000回振ったら
Aの領域に約5000回
Bの領域に約5000回と考える
(AとBの面積は同じ)
→面積に比例すると考えられる

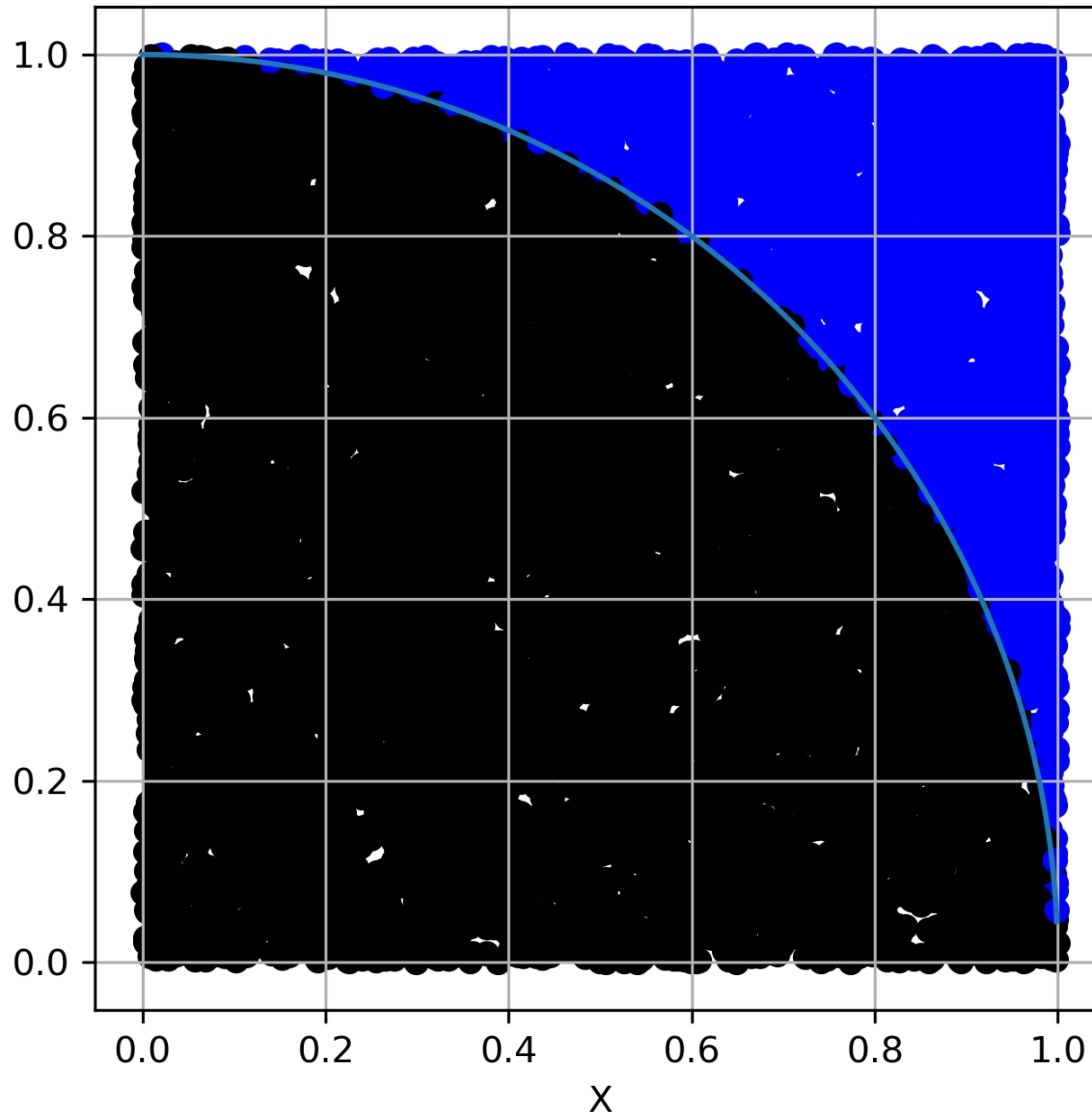


円の面積に比例するので
乱数を振った場合
面積側と面積の外の比は
 $\pi/4:1/4-\pi/4$ の割合で
になると予想される



正方形の面積は1なので
全体の個数を円の領域内の個数で割れば
面積が出てくる
 $1 \times \text{円の領域内の個数} / \text{全体の個数}$

モンテカルロ法(10000回試行)



```
# N回の試行にかかる時間を計測します
start_time = time.clock()
```

```
# N回の試行を行っています
for i in range(0, N):
    # 0から1の間で一様乱数を発生させ、変数
    score_xに格納してください
    score_x = np.random.rand()
    # 0から1の間で一様乱数を発生させ、変数
    score_yに格納してください
    score_y = np.random.rand()
    if score_x * score_x + score_y * score_y < 1:
        # 円内に入ったものは黒で表示させ、外
        れたものは青で表示させてください
        plt.scatter(score_x, score_y, marker='o',
        color='k')
        # 円内に入ったならば、上で定義した変
        数 X に 1 ポイント加算してください
        X = X + 1
    else:
        plt.scatter(score_x, score_y, marker='o',
        color='b')
```

```
# piの近似値をここで計算してください
```

モンテカルト法の問題点

- 収束するのが遅い。多大な計算をした割には答えがでるのが時間がかかる
- プログラムは容易

複数のグラフを描く(subplot)

```
%matplotlib inline
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
t = np.arange(0.0, 2.0, 0.01)    # 0～2.0の範囲で0.01刻
```

みの等差数列

```
s1 = np.sin(2*np.pi*t)          # (2・円周率・t)の正弦
```

```
s2 = np.sin(4*np.pi*t)          # (4・円周率・t)の正弦
```

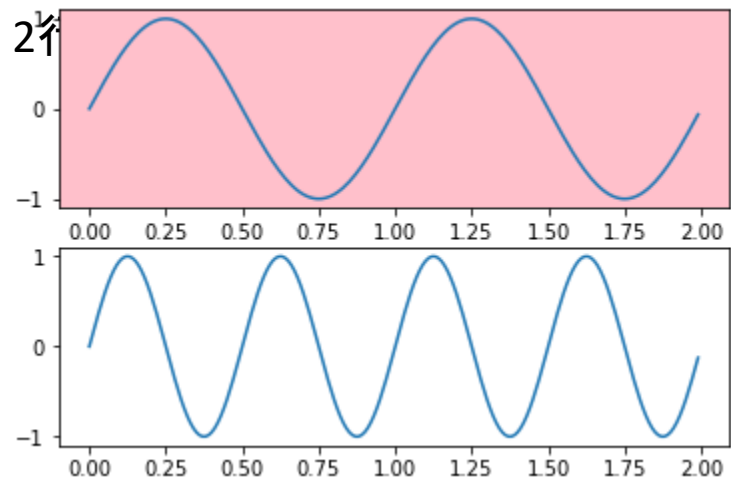
```
plt.subplot(211, facecolor='pink') # 2行×1列の上段を指定
```

```
plt.plot(t, s1)
```

```
plt.subplot(212, facecolor='white') # 2行×1列の下段を指定
```

```
plt.plot(t, s2)
```

```
plt.show()
```



subplot

- `plt.subplot(211, facecolor='pink')`
- 211 2行1列の行列の1番目を表す
- 211 2行1列の行列の2番目を表す

2行2列の複数のグラフを書く場合

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
t = np.arange(0.0, 2.0, 0.01)    # 0～2.0の範囲で0.01刻みの等差数列
s1 = np.sin(2*np.pi*t)          # (2・円周率・t)の正弦
s2 = np.sin(4*np.pi*t)          # (4・円周率・t)の正弦
s3 = np.sin(6*np.pi*t)          # (6・円周率・t)の正弦
s4 = np.sin(8*np.pi*t)          # (8・円周率・t)の正弦
```

```
plt.subplot(221, facecolor='pink') # 2行×2列の第1行、第1列を指定
plt.plot(t, s1)
plt.subplot(222, facecolor='white') # 2行×2列の第1行、第2列を指定
plt.plot(t, s2)
plt.subplot(223, facecolor='silver') # 2行×2列の第2行、第1列を指定
plt.plot(t, s3)
plt.subplot(224, facecolor='pink') # 2行×2列の第2行、第2列を指定
plt.plot(t, s4)
plt.show()
```

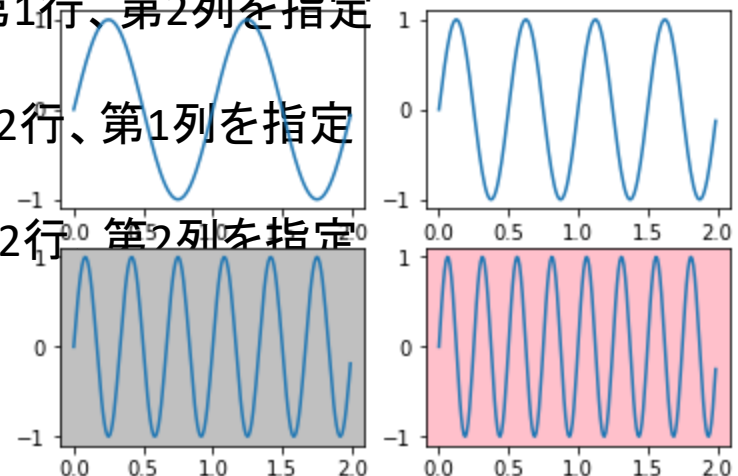


figure ,add_subplot

- 2行4列のグラフを作る

plt.subplots

```
import matplotlib.pyplot as plt
import numpy as np
x_list = range(0, 10)
y_list = x_list
np.random.randint(4, 10, (2, 1))
(fig, ax) = plt.subplots(2, 4, figsize = (10, 6)) # 2行4列 のグラフの作成 figsizeは表示の大きさ
fig.suptitle('Title')
ax[0, 0].bar(x_list, y_list)
ax[0, 1].plot(x_list, y_list, 'b')
ax[0, 2].plot(x_list, y_list, 'ro')
ax[0, 3].scatter(x_list, y_list)
ax[1, 0].bar(x_list, y_list) # 1つのグラフに重ねることもできる
ax[1, 0].plot(x_list, np.array(y_list))
ax[1, 0].scatter(x_list, np.array(y_list))
plt.bar(x_list, x_list) # 普通にプロット
plt.plot(x_list, y_list)
plt.show()
```

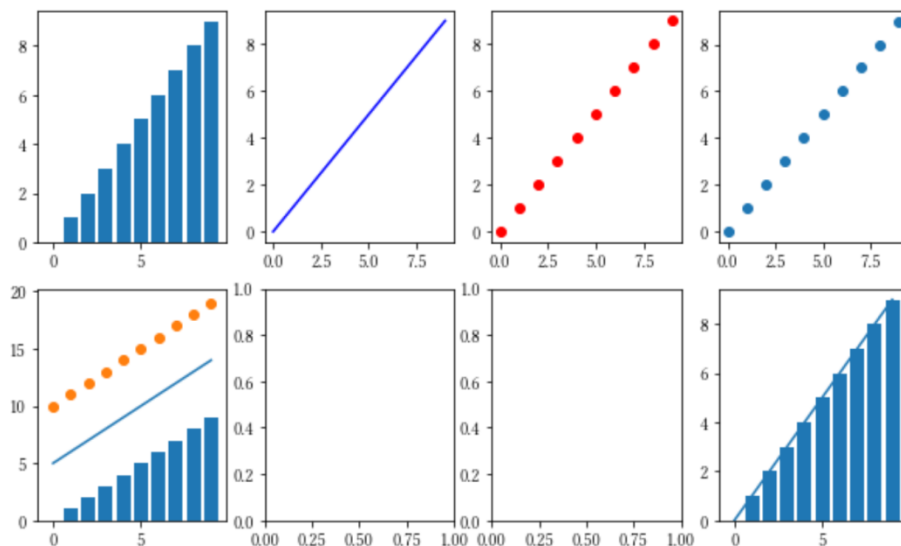


figure ,add_subplot

```
# サンプル4-10-1
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x1 = np.linspace(0.0, 5.0)      # 0.0～5.0の等差数列(要素数50)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1) # x1の減衰振動のシミュレーション
x2 = np.linspace(0.0, 3.0)      # 0.0～2.0の等差数列(要素数50)
y2 = np.cos(2 * np.pi * x2) * np.exp(-x1) # x2の減衰振動のシミュレーション

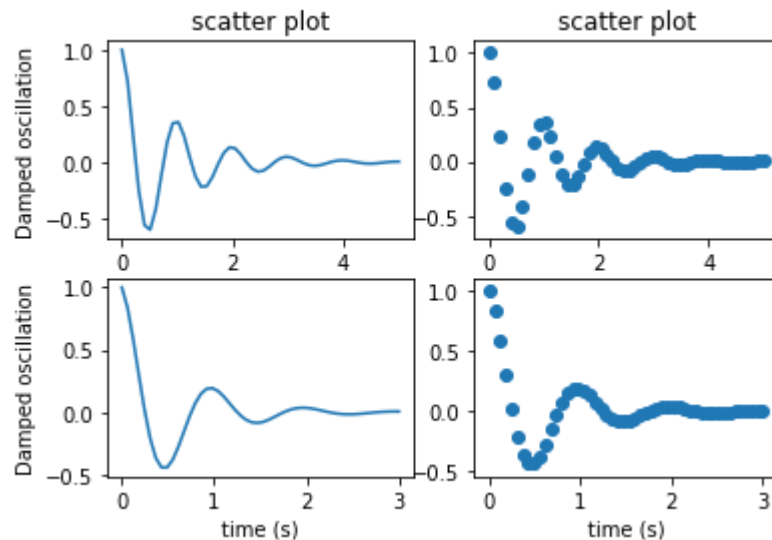
fig = plt.figure()             # Figureを生成
# 左上にx1、y1のラインをプロット
ax1 = fig.add_subplot(221)      # (221)にAxesを追加
ax1.plot(x1, y1)                # ラインをプロット
ax1.set_title('scatter plot')   # タイトル
ax1.set_ylabel('Damped oscillation') # y軸のラベル

# 右上にx1、y1のマーカーをプロット
ax2 = fig.add_subplot(222)      # (222)にAxesを追加
ax2.scatter(x1, y1, marker='o') # 散布図
ax2.set_title('scatter plot')   # タイトル

# 左下にx2、y2のラインをサブプロット
ax3 = fig.add_subplot(223)      # (223)にAxesを追加
ax3.plot(x2, y2)                # ラインをプロット
ax3.set_xlabel('time (s)'),      # x軸のラベル
ax3.set_ylabel('Damped oscillation') # y軸のラベル

# 右下にx2、y2のマーカーをサブプロット
ax4 = fig.add_subplot(224)      # (224)にAxesを追加
ax4.scatter(x2, y2, marker='o') # 散布図
ax4.set_xlabel('time (s)')       # x軸のラベル

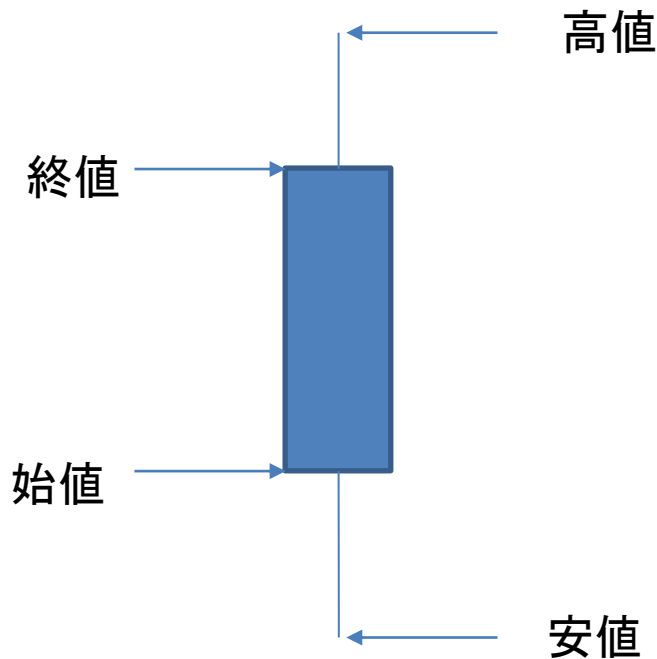
plt.show()
```



ローソク足

ローソク足

- 株価はその日の始値、高値、安値、終値を記したもの。



ファイルからローソク足を読み込んで
グラフを書く

```
#rosoku.py
```

```
import pandas_datareader as web
```

```
import numpy as np
```

```
import matplotlib as mpl
```

```
import matplotlib.pyplot as plt
```

```
import pandas as pd
```

```
import seaborn as sns
```

```
import mpl_finance as mpf
```

```
def candlewrite(fname,ax):
```

```
    nikkei=pd.read_csv(fname)
```

```
    df=nikkei[1:75].copy()
```

```
    mpf.candlestick2_ohlc(ax,df["Open"],df["High"],df["Low"],df["Close"],width=0.8,color='black')
```

```
    ax.plot(df['Close'].rolling(15).mean().values,label='rolling(10)')
```

```
    ax.plot(df['Close'].rolling(25).mean().values,label='rolling(15)')
```

```
    ax.grid(which='both')
```

```
    ax.grid(which='both')
```

```
    ax.legend()
```

```
(fig,ax)=plt.subplots(3,5,figsize=(12,6))
```

```
fig.suptitle('Title')
```

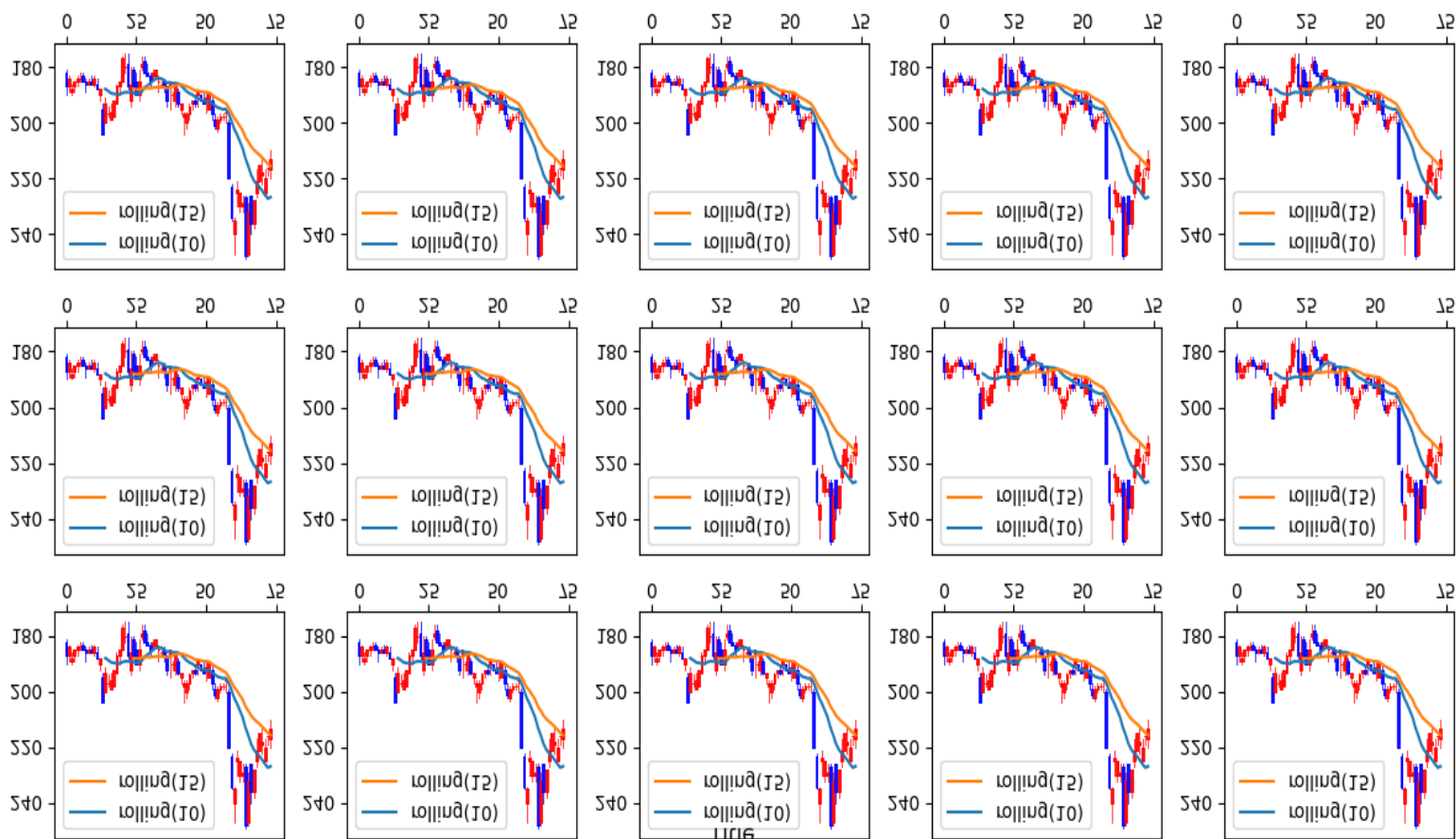
```
candlewrite('nikeikabuka.csv',ax[0,0])
```

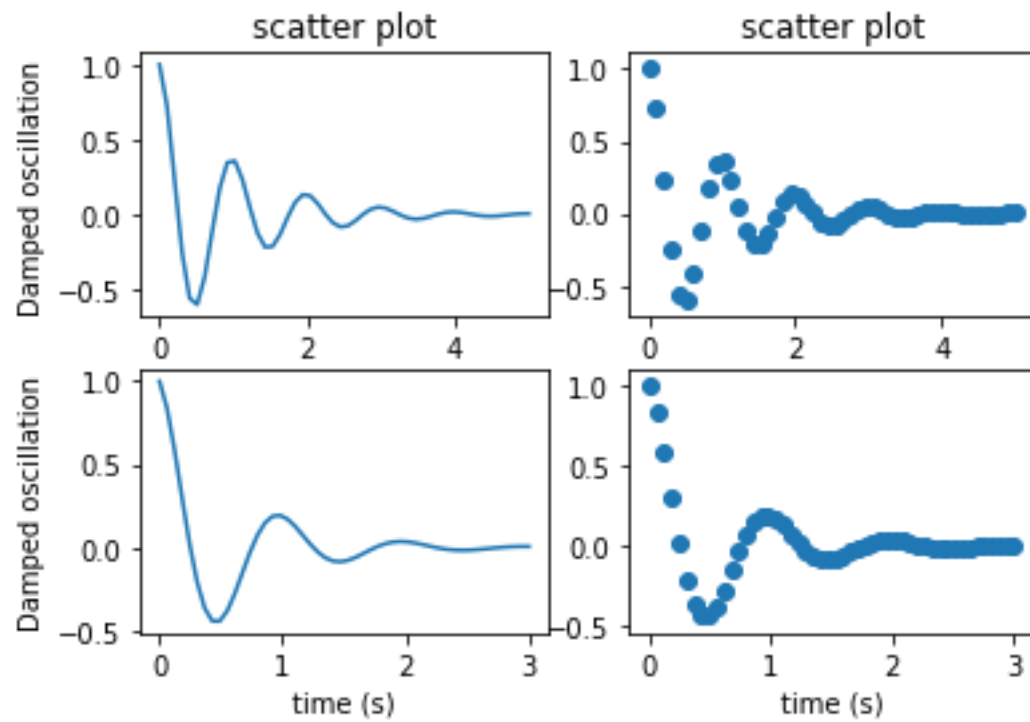
```
candlewrite('nikeikabuka.csv',ax[2,4])
```

```
plt.tight_layout()
```

```
plt.show()
```


結果





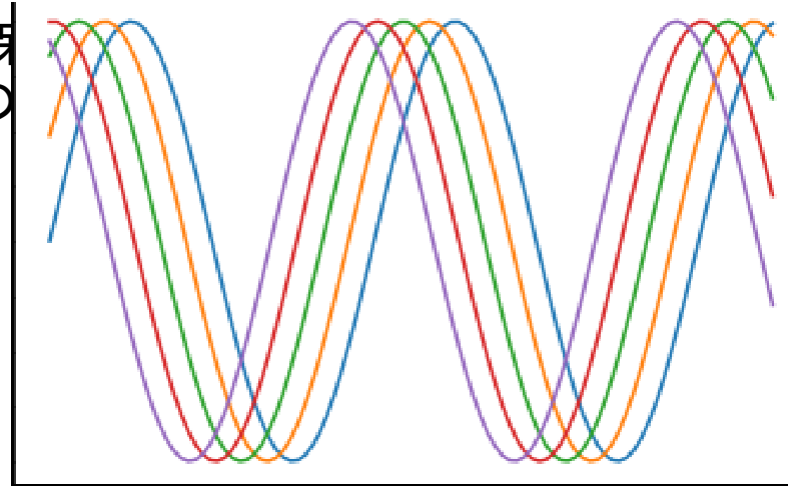
seaborn

正弦波

```
# サンプル3-06-2
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns # Seabornのインポート

sns.set()          # Seabornで出力できるようにする
x = np.linspace(0, 14, 100)
for i in range(5):
    plt.plot(x, np.sin(x + i*0.5))
    plt.savefig("sin.pdf")#この位置に保
    plt.savefig("sin.png",dpi=600)#この

plt.show()         # Seabornの
```



3Dグラフ

サンプル4-38-1

%matplotlib inline

import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import axes3d

```
def func(x,          # int: f(x,y)のxの値
        y          # int: f(x,y)のyの値
        ):
    return x**2 + y**2
```

```
return x**2 + y**2    # float: f(x, y)=x^2
```

```
x1 = np.arange(-3, 3, 0.25) # x_1軸を生
```

```
x2 = np.arange(-3, 3, 0.25) # x_2軸を生
```

```
X, Y = np.meshgrid(x1, x2) # 2次元の格
```

```
Z = func(X, Y)            # 関数f(x, y)に配列x,yを代入し、-3から
```

```
# 3までの0.25刻みのZ値のリストを取得
```

```
fig = plt.figure()        # Figureを生成
```

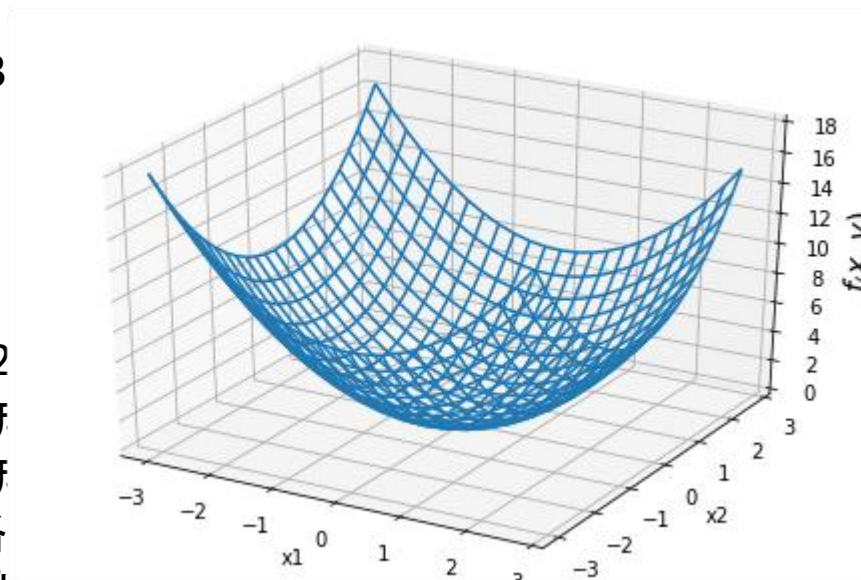
```
ax = axes3d.Axes3D(fig)   # Axes3Dオブジェクトを配置する
```

```
ax.set_xlabel("x1")       # x1の軸ラベル
```

```
ax.set_ylabel("x2")       # x2の軸ラベル
```

```
ax.set_zlabel(r"$f(x, y)$", size=15) # f(x1,x2)の軸ラベル
```

```
ax.plot_wireframe(X,Y,Z)   # x1、x2、f(x1,x2)の曲線をプロット
```



- ```

import sys,os
#sys.path.append(os.pardir)
import numpy as np
#from dataset.mnist import load_mnist
from keras.datasets import mnist
import keras
from keras.datasets import mnist
import numpy as np
from PIL import Image

def img_save(i,seikai,img):
 pil_img = Image.fromarray(np.uint8(img))
 pil_img.save(str(i)+'-'+str(seikai)+'-minitdata.png')

#(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)
(x_train, y_train), (x_test, y_test) = mnist.load_data()

for i in range(0,60000):
 print(i)
 img = x_train[i]
 label = y_train[i]
 print(label)
 print(img.shape)
 img = img.reshape(28, 28)
 img_save(i,label,img)

"""
for i in range(0,60000):
 img = x_train[i]
 label = t_train[i]
 print(label)
 print(img.shape)
 img = img.reshape(28, 28)
 print(img.shape)
 img_save(img)

```

- [返信](#)[返信](#) [転送](#)[転送](#) [移動](#)

# MINISTを使ったグラフ演習

# MNISTデータ構造

- ファイル
- MNIST データは、次の4つのファイルで構成されます。役割ごとにファイルが分かれています。
- train-images-idx3-ubyte: 学習用の画像セット
- train-labels-idx1-ubyte: 学習用のラベルセット
- t10k-images-idx3-ubyte: 検証用の画像セット
- t10k-labels-idx1-ubyte: 検証用のラベルセット



| offset | type           | value            | description |
|--------|----------------|------------------|-------------|
| 0000   | 32 bit integer | 0x00000801(2049) | 識別子(定数)     |
| 0004   | 32 bit integer | 60000 or 10000   | ラベルデータの数    |
| 0008   | unsigned byte  | 0 ~ 9            | 1つ目のデータのラベル |
| 0009   | unsigned byte  | 0 ~ 9            | 2つ目のデータのラベル |
| ....   | ....           | ....             | ....        |
| xxxx   | unsigned byte  | 0 ~ 9            | 最後のデータのラベル  |

| offset | type           | value            | description      |
|--------|----------------|------------------|------------------|
| 0000   | 32 bit integer | 0x00000803(2051) | 識別子(定数)          |
| 0004   | 32 bit integer | 60000            | 画像データの数          |
| 0008   | 32 bit integer | 28               | 1画像あたりのデータ行数     |
| 0012   | 32 bit integer | 28               | 1画像あたりのデータ列数     |
| 0016   | unsigned byte  | 0 ~ 255          | 1つめの画像の1ピクセル目の値  |
| 0017   | unsigned byte  | 0 ~ 255          | 1つめの画像の2ピクセル目の値  |
| ....   | ....           | ....             | ....             |
| xxxx   | unsigned byte  | 0 ~ 255          | 最後の画像の784ピクセル目の値 |

# MNISTを画像にして保存する(mnist.py)

```
#-----
#Ministデータをディレクトリmnistに格納する
#データは番号+正解の数字+mnistで格納される
#-----
import sys,os
import numpy as np
from keras.datasets import mnist
import keras
from keras.datasets import mnist
import numpy as np
from PIL import Image
def img_save(i,seikai,img):
 path=u'¥¥mnist¥¥'
 pil_img = Image.fromarray(np.uint8(img))
 pil_img.save(path+str(i)+'-'+str(seikai)+'-minitdata.png')

(x_train, y_train), (x_test, y_test) = mnist.load_data()

for i in range(0,60000):
 img = x_train[i]
 label = y_train[i]
 img = img.reshape(28, 28)
 img_save(i,label,img)
```

# MINITをmatplotlibで表示する

# 必要なライブラリのインポート

```
import keras
```

```
from keras.datasets import mnist
```

# Jupyter notebookを利用している際に、notebook内にplot結果を表示するようにする

```
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

#Kerasの関数でデータの読み込み。データをシャッフルして学習データと訓練データに分割

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

#MNISTデータの表示

```
fig = plt.figure(figsize=(9, 9))
```

```
fig.subplots_adjust(left=0, right=1, bottom=0, top=0.5,
hspace=0.05, wspace=0.05)
```

```
for i in range(81):
```

```
 ax = fig.add_subplot(9, 9, i + 1, xticks=[], yticks=[])
```

```
 ax.imshow(x_train[i].reshape((28, 28)), cmap='gray')
```

# 課題1

- MNISTのデータ60000枚の文字の種類をカウントして数を棒グラフにしてみよう

# 19章 深層学習の実践

```
x_test = x_test.reshape(x_test.shape[0], 784)[1000:]
y_train = to_categorical(y_train)[:6000]
y_test = to_categorical(y_test)[:1000]
```

## 19.1.1

```
model = Sequential()
model.add(Dense(256, input_dim=784))
model.add(Activation("sigmoid"))
model.add(Dense(128))
model.add(Activation("sigmoid"))
model.add(Dropout(rate=0.5))
model.add(Dense(10))
model.add(Activation("softmax"))
```

```
sgd = optimizers.SGD(lr=0.1)
model.compile(optimizer=sgd, loss="categorical_crossentropy",
metrics=["accuracy"])
```

# epochs 数は 5 を指定します

```
history = model.fit(X_train, y_train, batch_size=500, epochs=5,
verbose=1, validation_data=(X_test, y_test))
```

# acc、val\_acc のプロットです

```
plt.plot(history.history["accuracy"], label="acc", ls="-", marker="o")
plt.plot(history.history["val_accuracy"], label="val_acc", ls="-",
marker="x")
plt.ylabel("accuracy")
```

# 深層学習とは

- 機械学習の一手法
- 脳の神経ネットワークを模倣

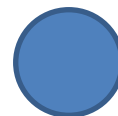
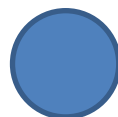
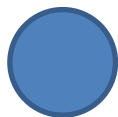
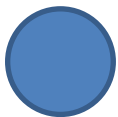
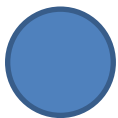


# 深層学習2

- $X1$ と $W1$ をかけて $X2$ と $W2$ をかけたものを足して $\theta$ を足したものを出力 $y$ とする

# 深層学習が注目されるようになった理由

- (1)膨大なデータが手に入るようになった  
(ビックデータ)
- (2)コンピューターの計算環境が整い始めた  
(AWSのクラウドやGPUなど)
- (3)従来、精度が上がらないと考えられていた  
ニューラルネットワークが多数重ねて(深い層)  
処理すると性能が上がるとうわかり始めた



# 手書き数字分類

- データを用意
- ニューラルネットワークモデルの構築
- モデルにデータを与えて学習させる
- モデルの分類精度を評価

4



# Kerasの導入

- KerasはTensorflowのラッパー
- Tensorflowより簡潔にコードを書くことができる
- Pythonで書かれたオープンソースニューラルネットワークライブラリ
- tensorflowはgoogleが開発

# コード解説

```
(X_train, y_train), (X_test, y_test) =
mnist.load_data()
```

データをministデータをWebからダウンロードする。1度データを読み込むと2回目からはダウンロードしたデータを使うのでダウンロードしなくなる仕組み



# X\_train, y\_train, X\_test, y\_test

- X\_train, y\_trainは訓練用データ(6万枚)  
X\_trainは6万枚のデータで $28 \times 28$ の大きさ  
y\_trainはデータ6万枚のデータの文字のラベル  
(0から9までの数字のラベル)
- X\_test, y\_testはテストデータ(1万枚)  
上に同様

# 28 × 28に変更

- `print(X_train[0].reshape(28,28))`
- `print( y_train[0])`

```
print(y_train[0])
```

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 3 18 18 18 126 136
 175 26 166 255 247 127 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 30 36 94 154 170 253 253 253 253
225 172 253 242 195 64 0 0 0 0]
 [0 0 0 0 0 0 0 49 238 253 253 253 253 253 253 253 253 251
93 82 82 56 39 0 0 0 0 0]
 [0 0 0 0 0 0 0 18 219 253 253 253 253 253 198 182 247 241
 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 80 156 107 253 253 205 11 0 43 154
 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 14 1 154 253 90 0 0 0 0
 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 139 253 190 2 0 0 0
 0 0 0 0 0 0 0 0 0 0]
```

# モデルの作成

# モデル部分のコード

```
model = Sequential()
model.add(Dense(256, input_dim=784))
model.add(Activation("sigmoid"))
model.add(Dense(128))
model.add(Activation("sigmoid"))
model.add(Dropout(rate=0.5))
model.add(Dense(10))
model.add(Activation("softmax"))
```

```
model.add(Dense(256,
 input_dim=784))
```

- ユニット数を128全結合

```
model.add(Activation("sigmoid"))
```

```
model.add(Dropout(rate=0.5))
```

```
model.add(Dense(10))
```



```
model.add(Activation("softmax"))
```

- シグモイド関数を使う

```
sgd = optimizers.SGD(lr=0.1)
```

- 最適化はSGDを使う
- 損失関数は交差エントロピーを使う



# 21 深層学習画像認識

- 21.1 深層学習画像認識
  - 21.1.1 画像認識
- 21.2 CNN
  - 21.2.1 CNNの概要
  - 21.2.2 畳み込み層
  - 21.2.3 プーリング層
  - 21.2.4 CNNの実装
  - 21.2.5 CNNを用いた分類(MNIST)
  - 21.2.6 CNNを用いた分類(cifar10)
- 21.3 ハイパーパラメータ
  - 21.3.1 filters (Conv層)
  - 21.3.2 kernel\_size (Conv層)
  - 21.3.3 strides (Conv層)
  - 21.3.4 padding (Conv層)
  - 21.3.5 pool\_size (Pool層)
  - 21.3.6 strides (Pool層)
  - 21.3.7 padding (Pool層)

## 21.1.1 画像認識

- 人間の脳視覚野と似た構造を持つ畳み込み層を使って特徴抽出を行う
- 全結合と違い二次元な特徴を抽出するのに優れている

## 21.2.1 CNNの概要

## 21.2.2 畳み込み層

## 21.2.3 プーリング層



## 21.2.4 CNNの実装

## 21.2.5 CNNを用いた分類(MNIST)

## 21.2.6 CNNを用いた分類(cifar10)

## 21.3 ハイパーパラメータ

## 21.3.1 filters (Conv層)

## 21.3.2 kernel\_size (Conv層)

## 21.3.3 strides (Conv層)

## 21.3.4 padding (Conv層)



## 21.3.5 pool\_size (Pool層)

## 21.3.6 strides (Pool層)

## 21.3.7 padding (Pool層)