

# 医療とAI・ビッグデータ応用

## ②MNISTの読み込みと前処理

本スライドは、自由にお使いください。  
使用した場合は、このQRコードからアンケート  
に回答をお願いします。



統合教育機構  
須藤毅顕

```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

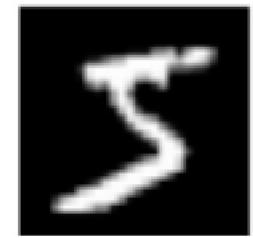
60000個

60000個

10000個

10000個

mnistのdataを読み込む



5



0



4



1

⋮



6



8

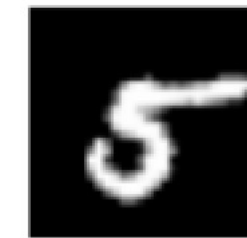


7



2

⋮



5



6

x\_train : 60000枚の画像の配列データ  
y\_train : 60000枚の正解の数字の配列データ  
x\_test : 10000枚の画像の配列データ  
y\_test : 10000枚の正解の数字の配列データ

# 画像を描画する

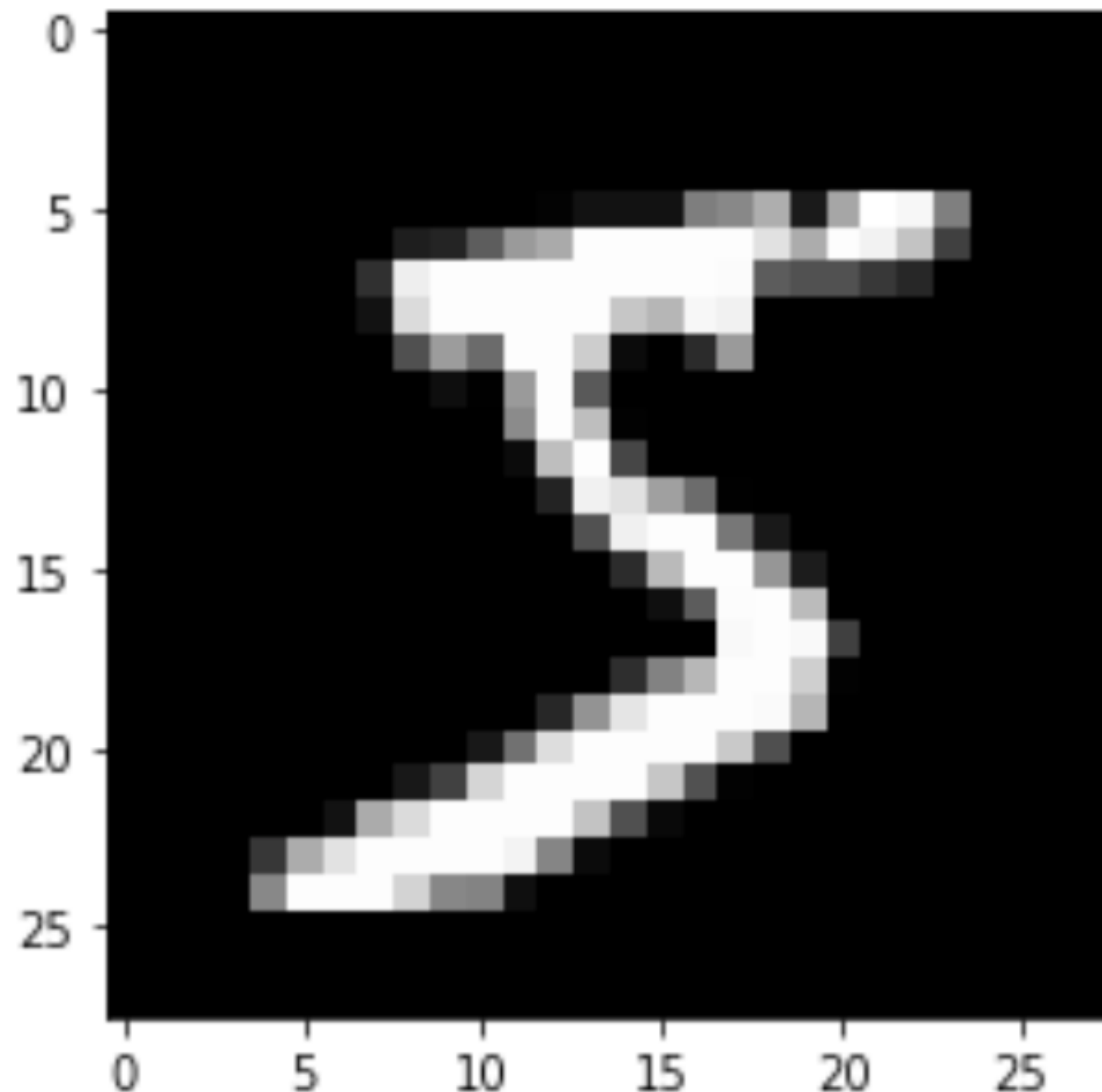
```
import matplotlib.pyplot as plt
plt.imshow(x_train[0], 'gray')
plt.show()
```

matplotlib(描画ライブラリ)

plt.imshow(画像もしくは配列, 'color\_mode')

'gray'で白黒を指定

plt.show()で表示



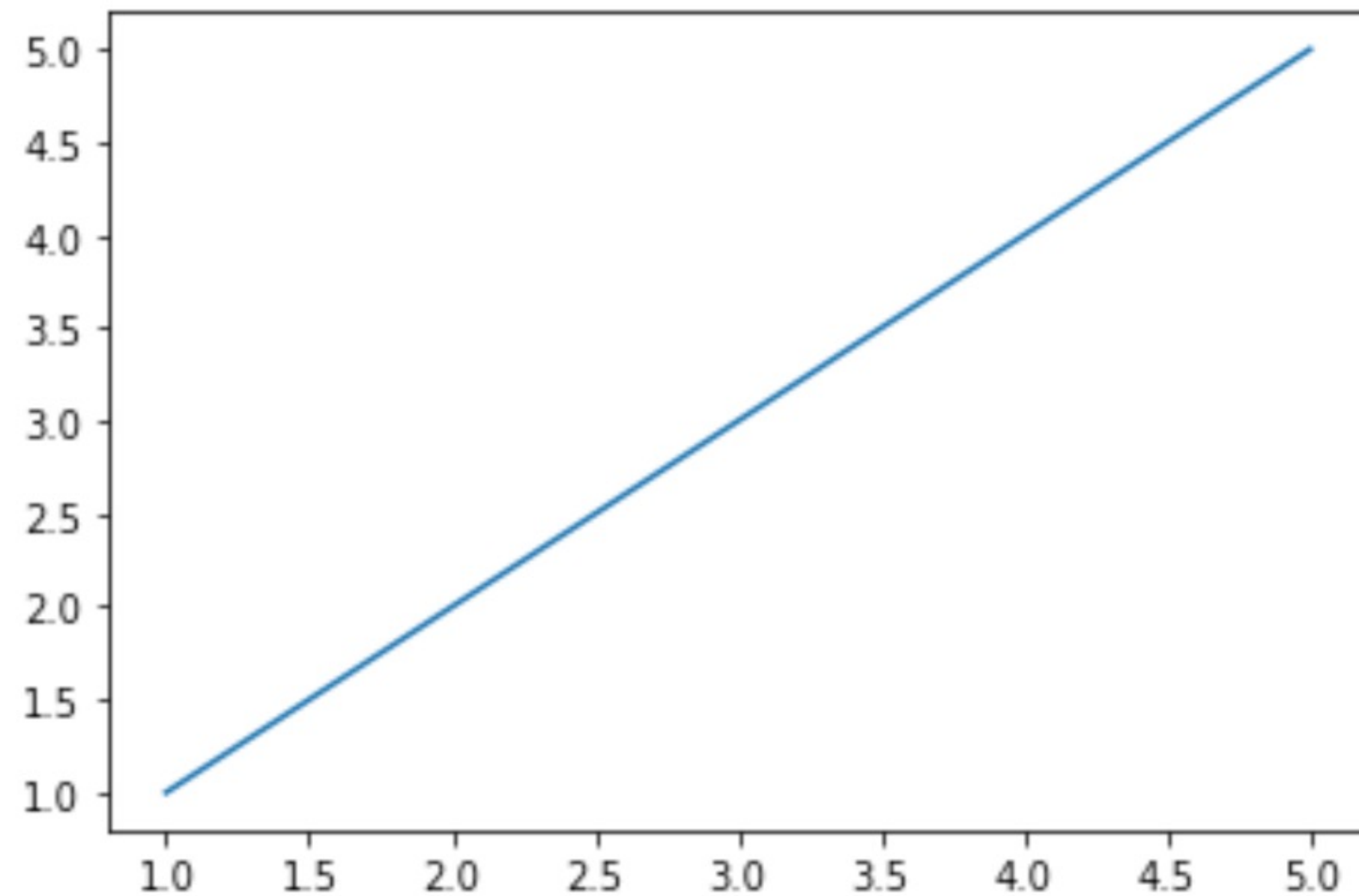
# 1つ目を取り出してみる

```
print(x_train[0])
```

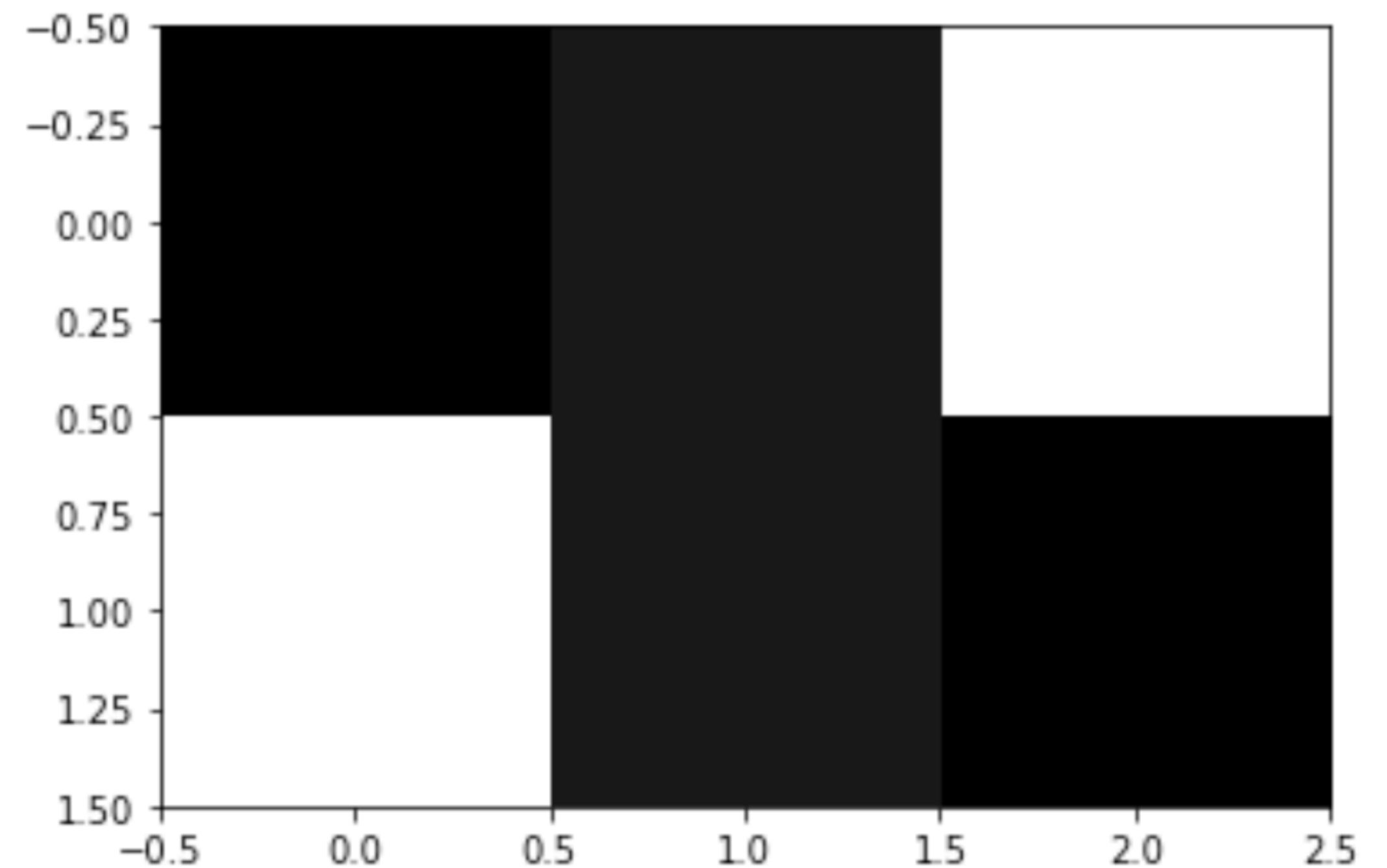
[illegible]

plt.plot(x,y)でxとyの値を直線で結ぶ  
plt.imshow(x)でxの画像データもしくは配列を描画する  
白黒(gray)を指示した場合、数字が大きいほど白い(0~255)

```
x = [1,2,3,4,5]  
y = [1,2,3,4,5]  
plt.plot(x,y)  
plt.show()
```

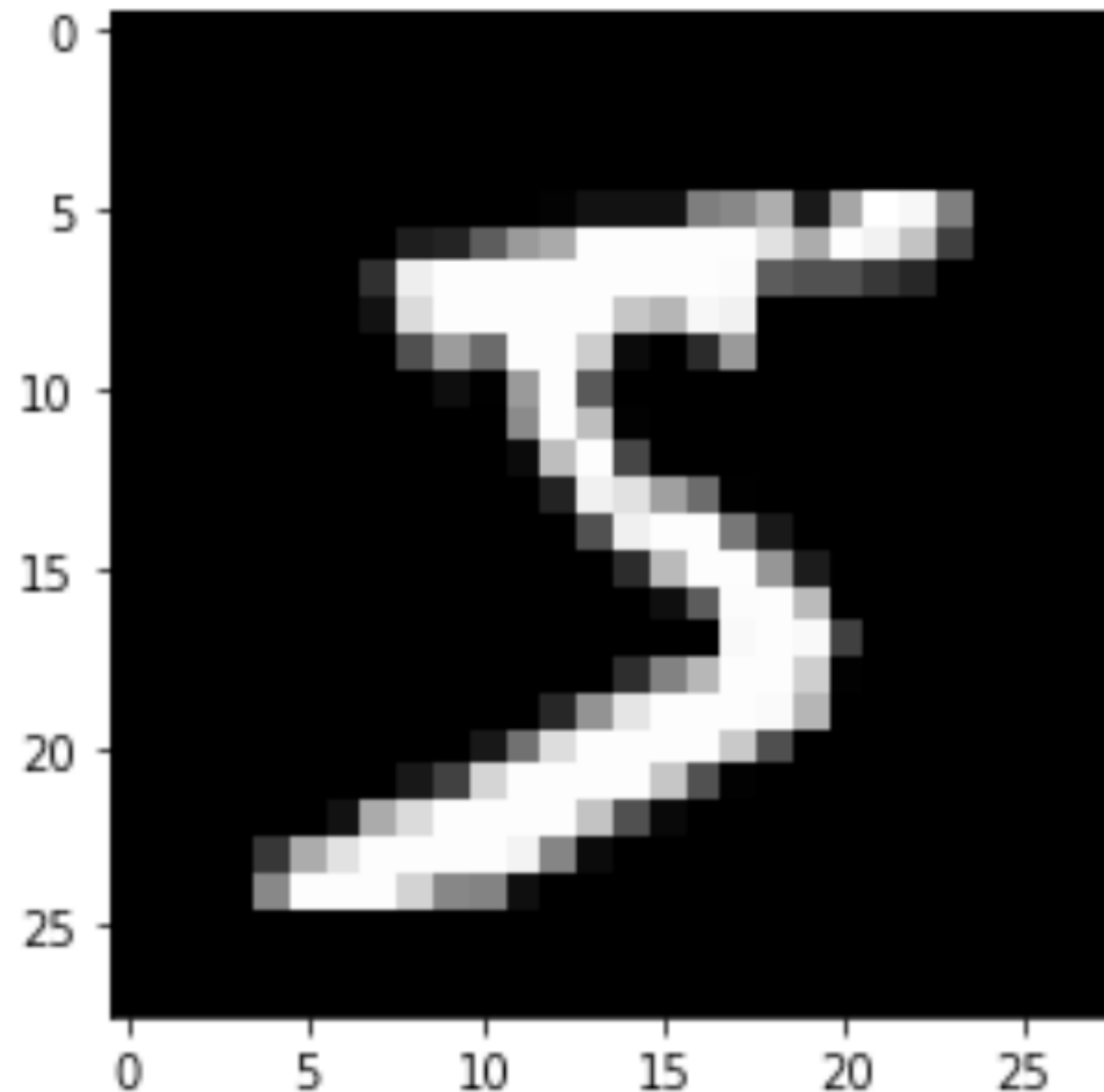


```
import numpy as np  
x = np.array([[1,10,100],[100,10,1]])  
plt.imshow(x, 'gray')  
plt.show()
```





```
import matplotlib.pyplot as plt
plt.imshow(x_train[0], 'gray')
plt.show()
```



画像を描画する

matplotlib(描画ライブラリ)

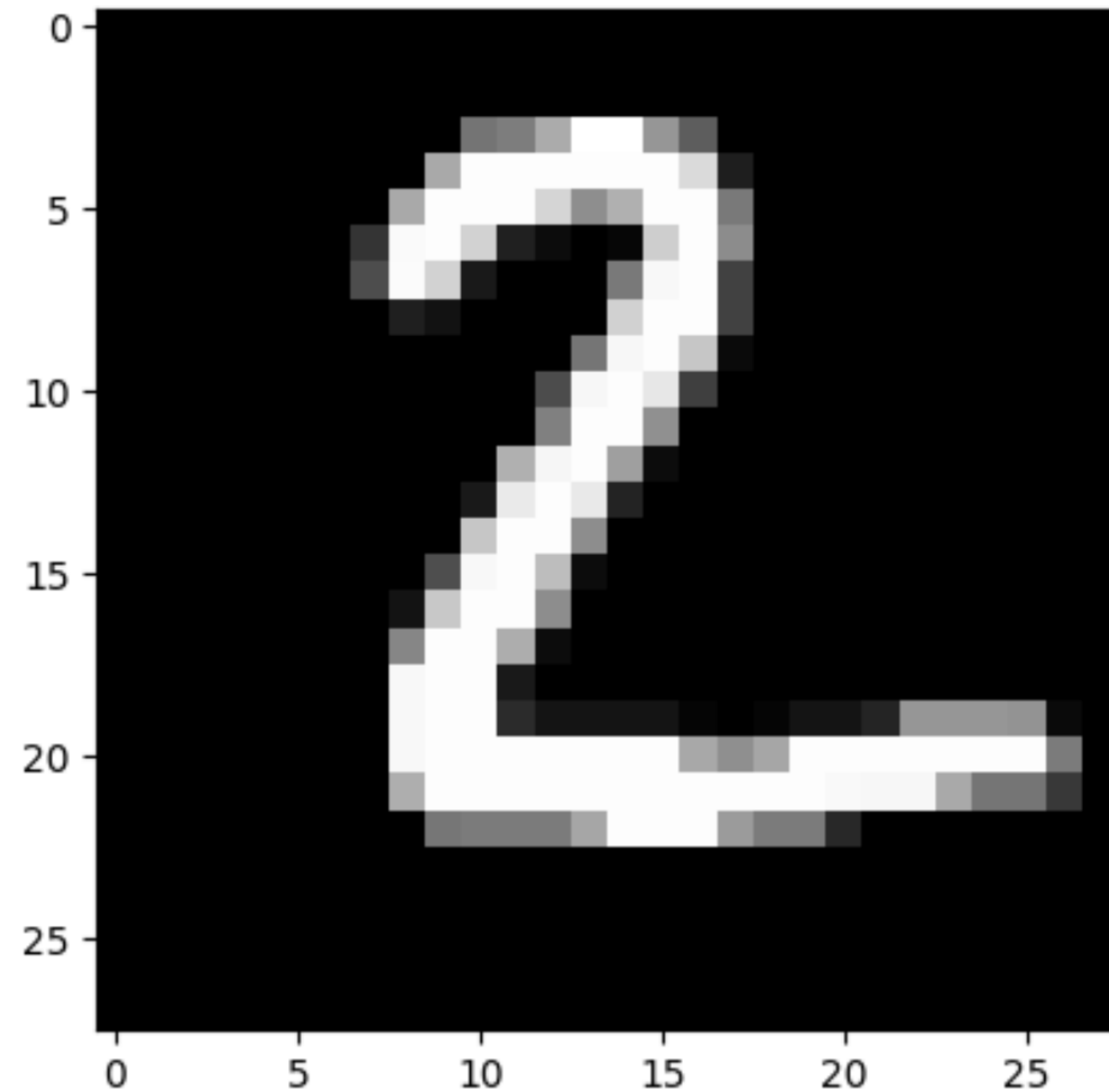
‘gray’で白黒を指定

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

5

数字の5らしい

```
import matplotlib.pyplot as plt
plt.imshow(x_test[1], 'gray')
plt.show()
```



画像を描画する

matplotlib(描画ライブラリ)

‘gray’で白黒を指定

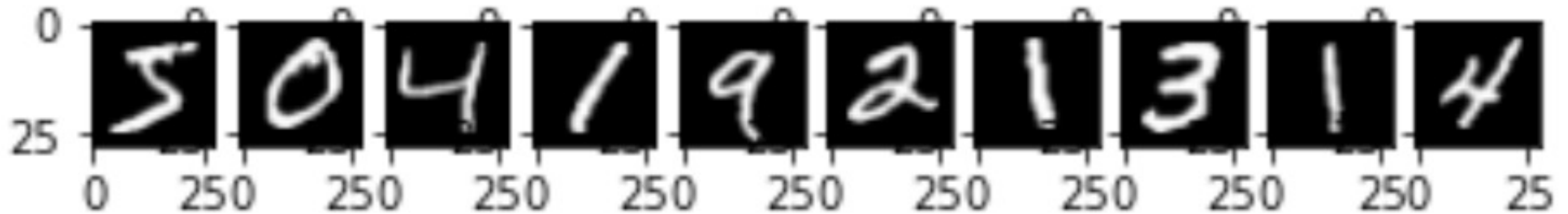
```
print(y_test[1])
```

2

数字の2らしい

10個並べてみる

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```





## for文とrange関数

```
for i in range(1,10,2):  
    print(i)
```

1  
3  
5  
7  
9

```
for i in range(5):  
    print(i)
```

0  
1  
2  
3  
4

=

```
for i in range(0,5,1):  
    print(i)
```

0  
1  
2  
3  
4

for (変数) in range(A,B,C):  
 (処理内容)

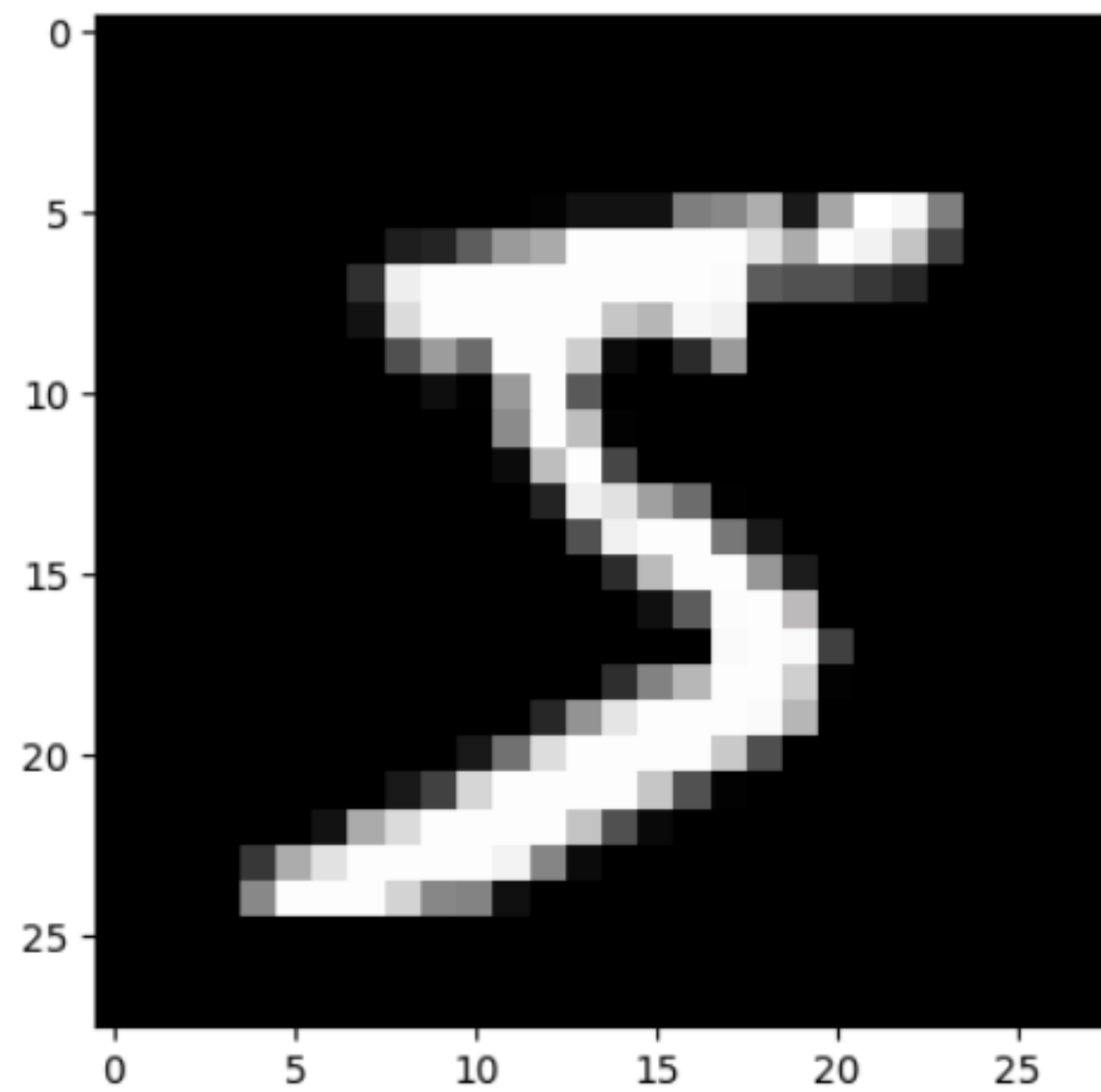
A(以上)からB(未満)でC刻みに変数に代入

この例だと1から1つ飛ばしで9まで  
iは順に1,3,5,7,9が代入されて処理

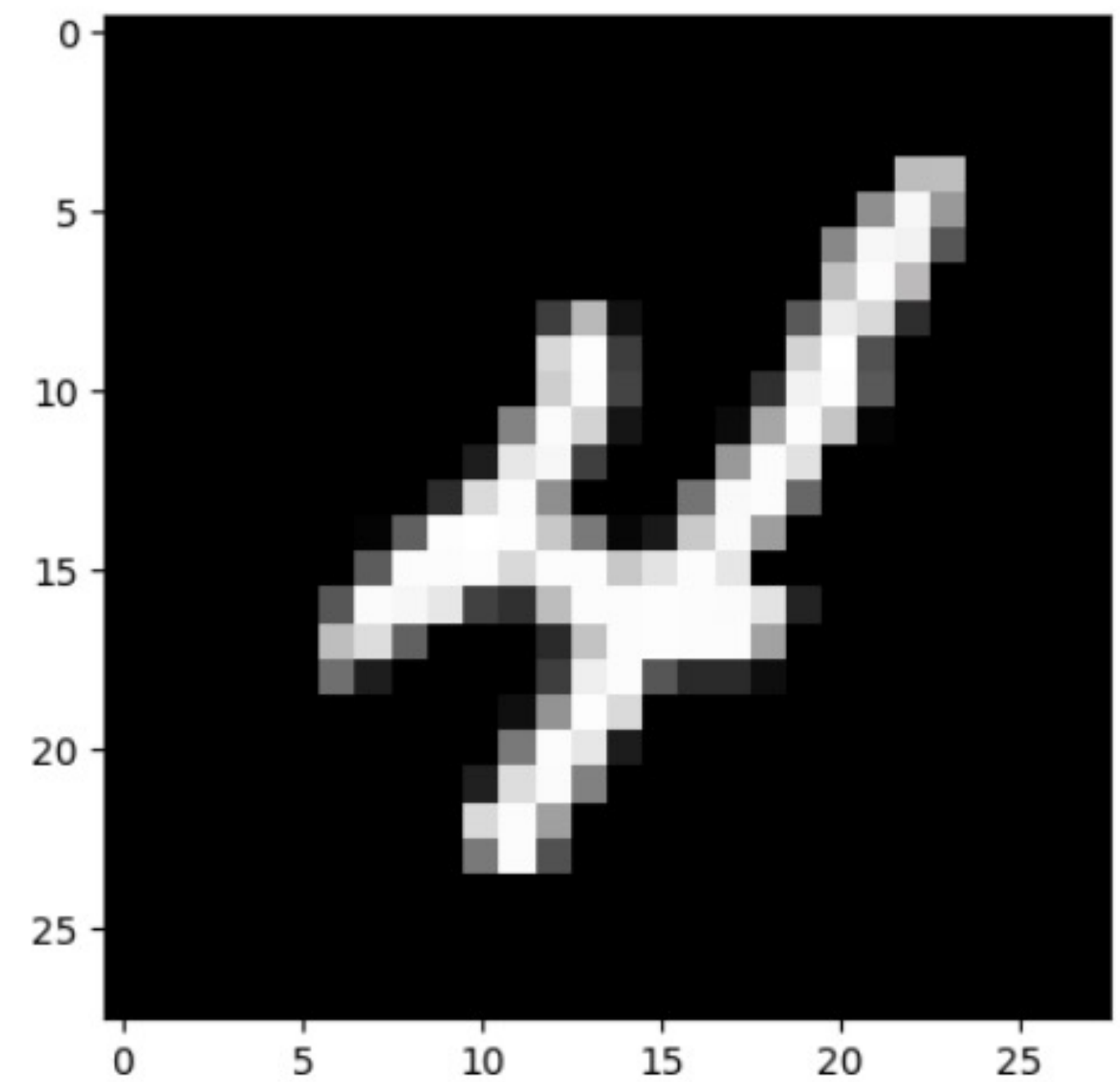
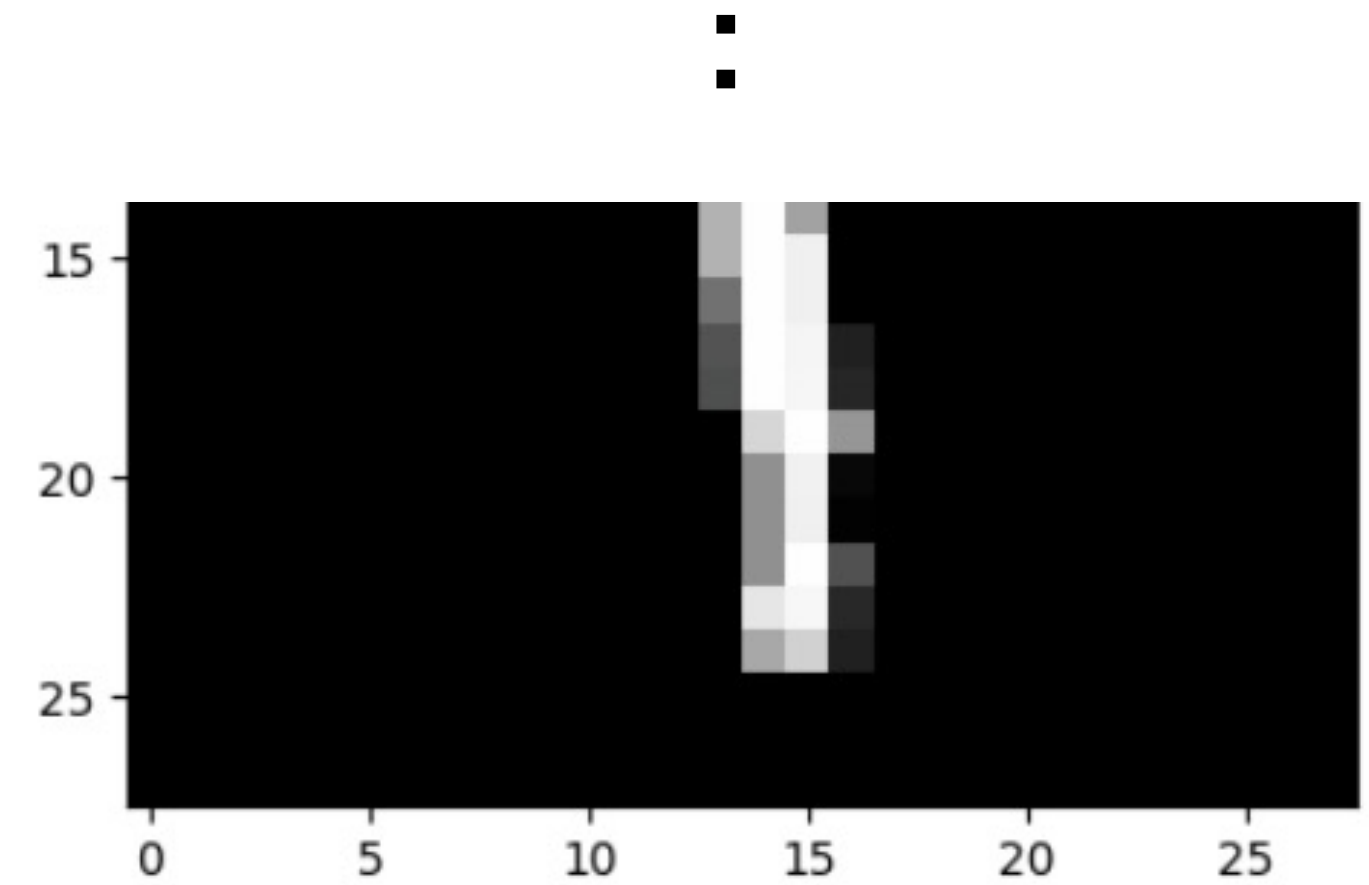
range(B)のように数字1つにすると  
開始のAを0、刻み幅Cを1の設定で省略出来る

for文

```
for i in range(10):  
    plt.imshow(x_train[i], 'gray')  
    plt.show()
```



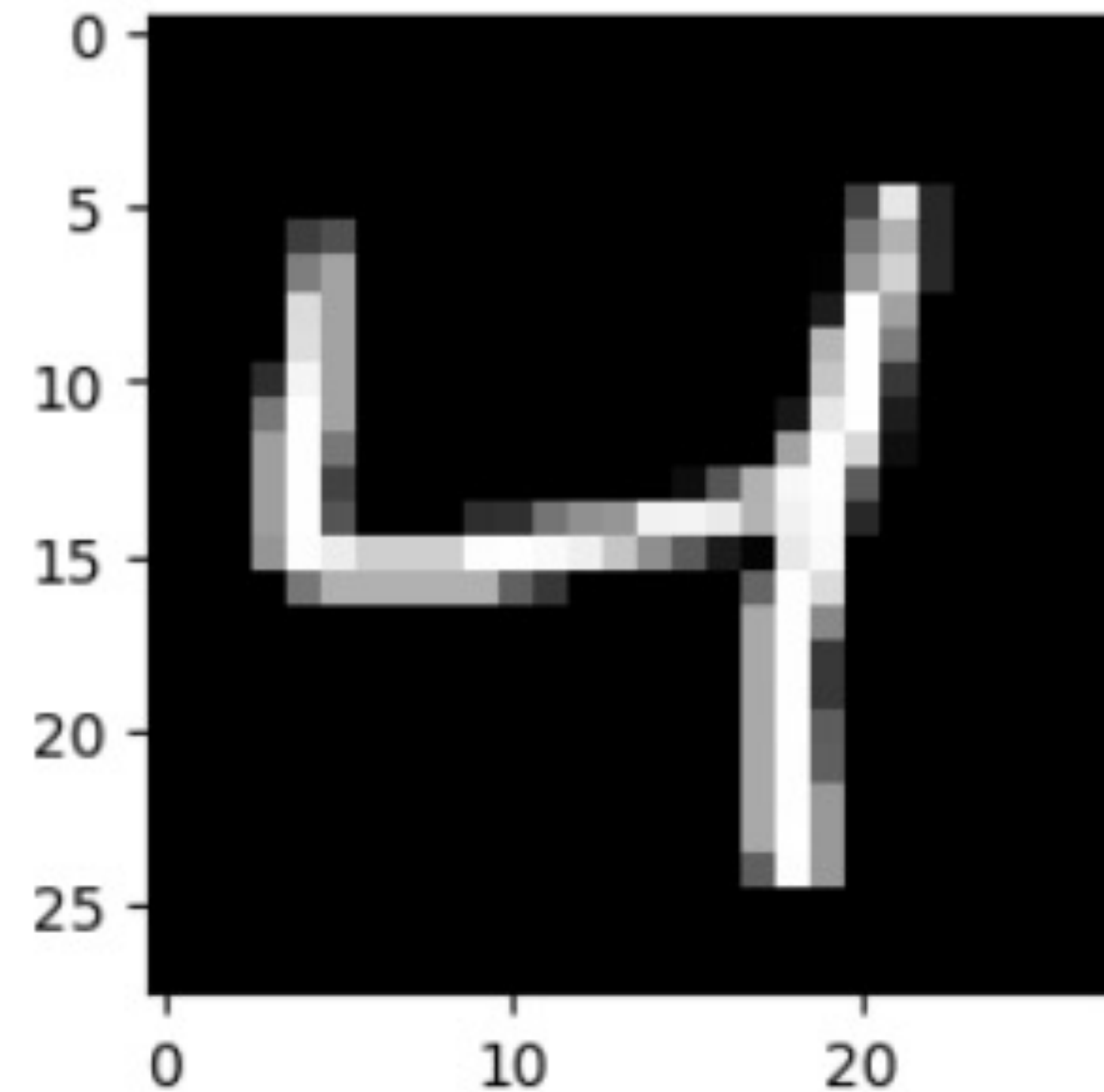
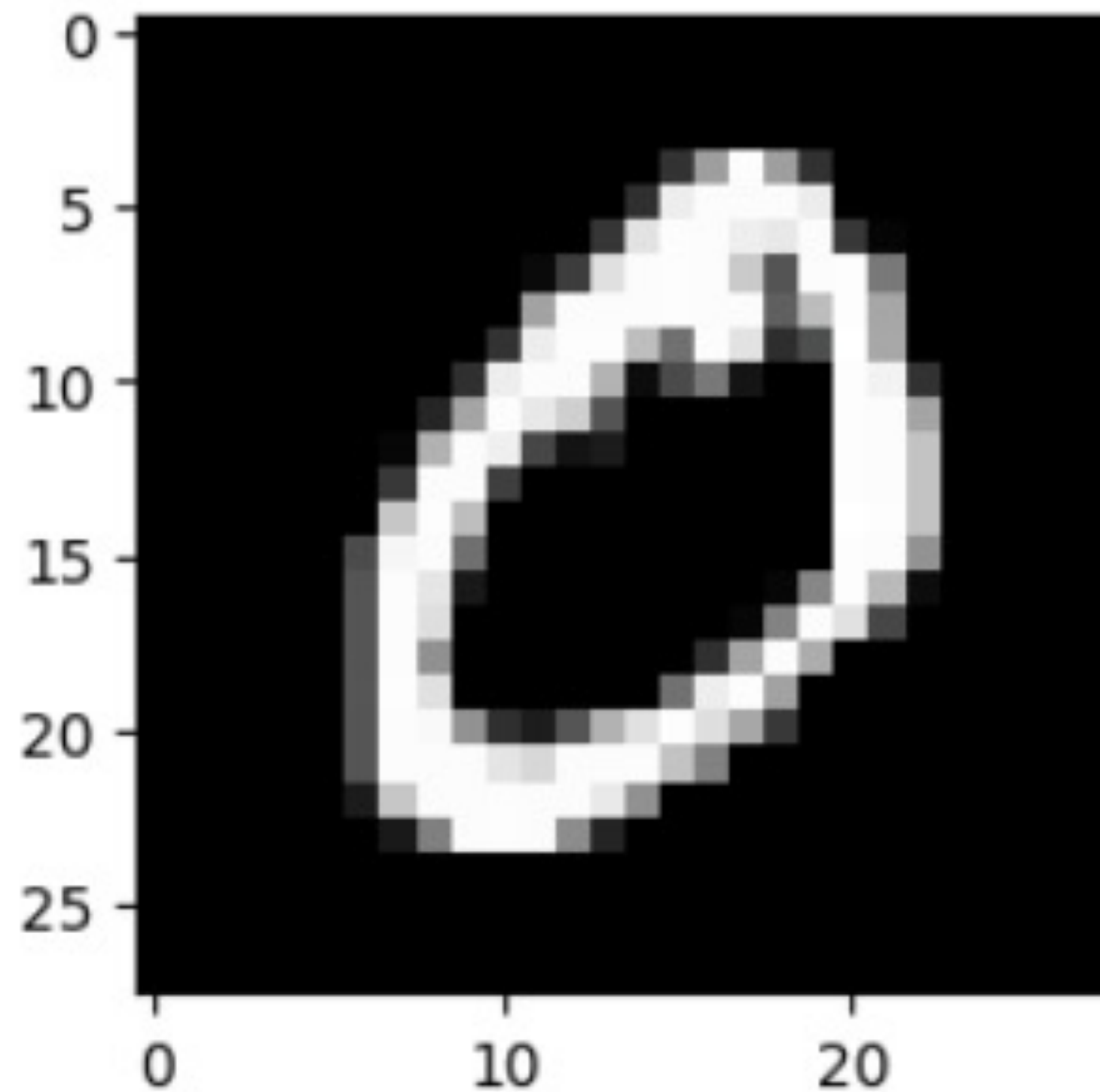
■  
■



plt.subplotは図を並べる plt.subplot(縦,横,左上から何番目か)

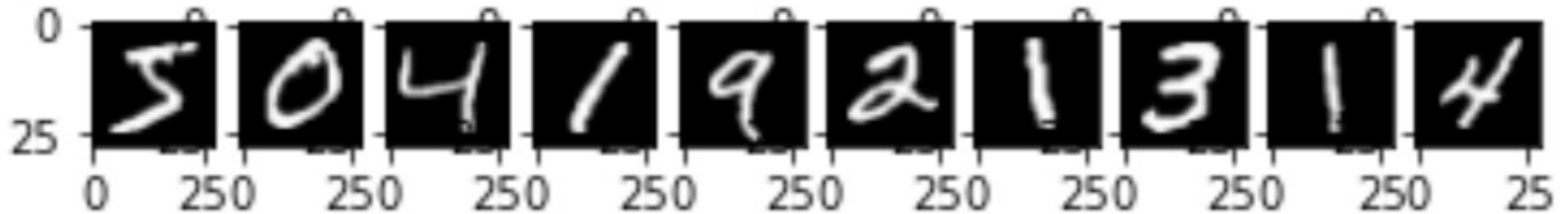
```
[8] plt.subplot(1,2,1)          ←縦1、横2に並べる内の1つ目の宣言
     plt.imshow(x_train[1], 'gray')
     plt.subplot(1,2,2)          ←縦1、横2に並べる内の2つ目の宣言
     plt.imshow(x_train[2], 'gray')

     plt.show()                  ←最後に図を表示する
```



10個並べてみる

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```



## for文

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```

plt.subplot(1,10,i+1)  
plt.imshow(x\_train[i], 'gray')

縦に1つ、横に10個、図を書く。  
i=0なので(1,10,1)で1番左の図を指定する



x\_train[0]



## for文

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```

plt.subplot(1,10,i+1)  
plt.imshow(x\_train[i], 'gray')

次がi=1  
(1,10,2)で左から2つ目の図を指定する  
plt.imshow(x\_train[1], 'gray')



## for文

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```

plt.subplot(1,10,i+1)  
plt.imshow(x\_train[i], 'gray')

最後はi=9  
(1,10,10)で左から10個目の図を指定する  
plt.imshow(x\_train[9], 'gray')



for文

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```

plt.subplot(1,10,i+1)  
plt.imshow(x\_train[i], 'gray')

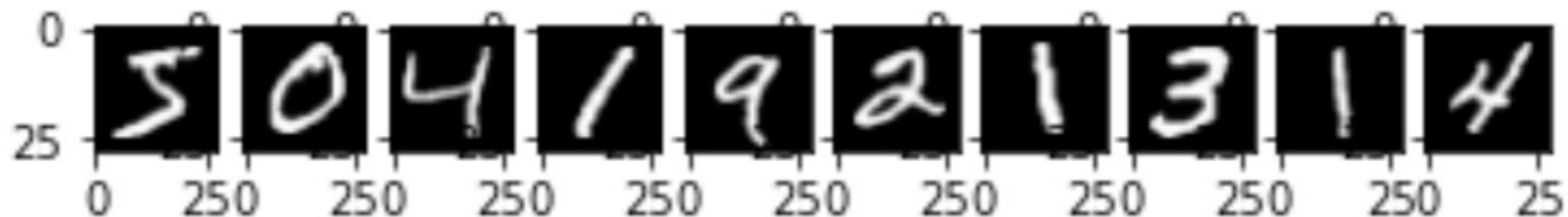
最後はi=9

(1,10,10)で左から10個目の図を指定する

plt.imshow(x\_train[1], 'gray')

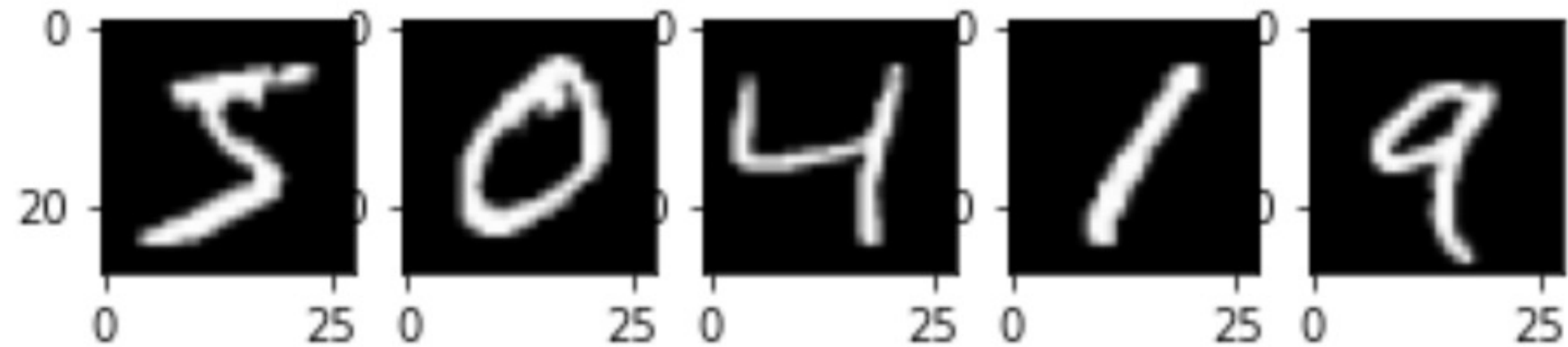


x\_train[9]



plt.subplot(2,5,i+1)にすると縦2、横5の図になる

```
for i in range(10):  
    plt.subplot(2,5,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```



正解も10個並べてみる

```
print(y_train[0:10])
```

```
[5 0 4 1 9 2 1 3 1 4]
```

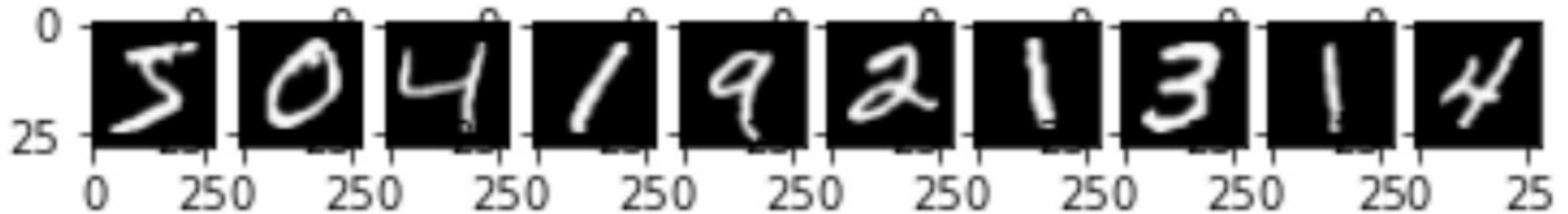
配列は[始まりの数字：終わりの数字]で中身(要素)を取り出せる

[0:10]で0から9番目まで！



x\_trainとy\_trainが特徴量と正解の関係になっている（図でも確認）

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```



```
print(y_train[0:10])
```

```
[5 0 4 1 9 2 1 3 1 4]
```

# 深層学習前のデータの整理

**x\_train (特徴量)**

- 画像の2次元の配列を1次元にする
- 正規化する

**y\_train (正解)**

- one-hot encoding

# 深層学習前のデータの整理

x\_train (特徴量)

- ・ 画像の2次元の配列を1次元にする

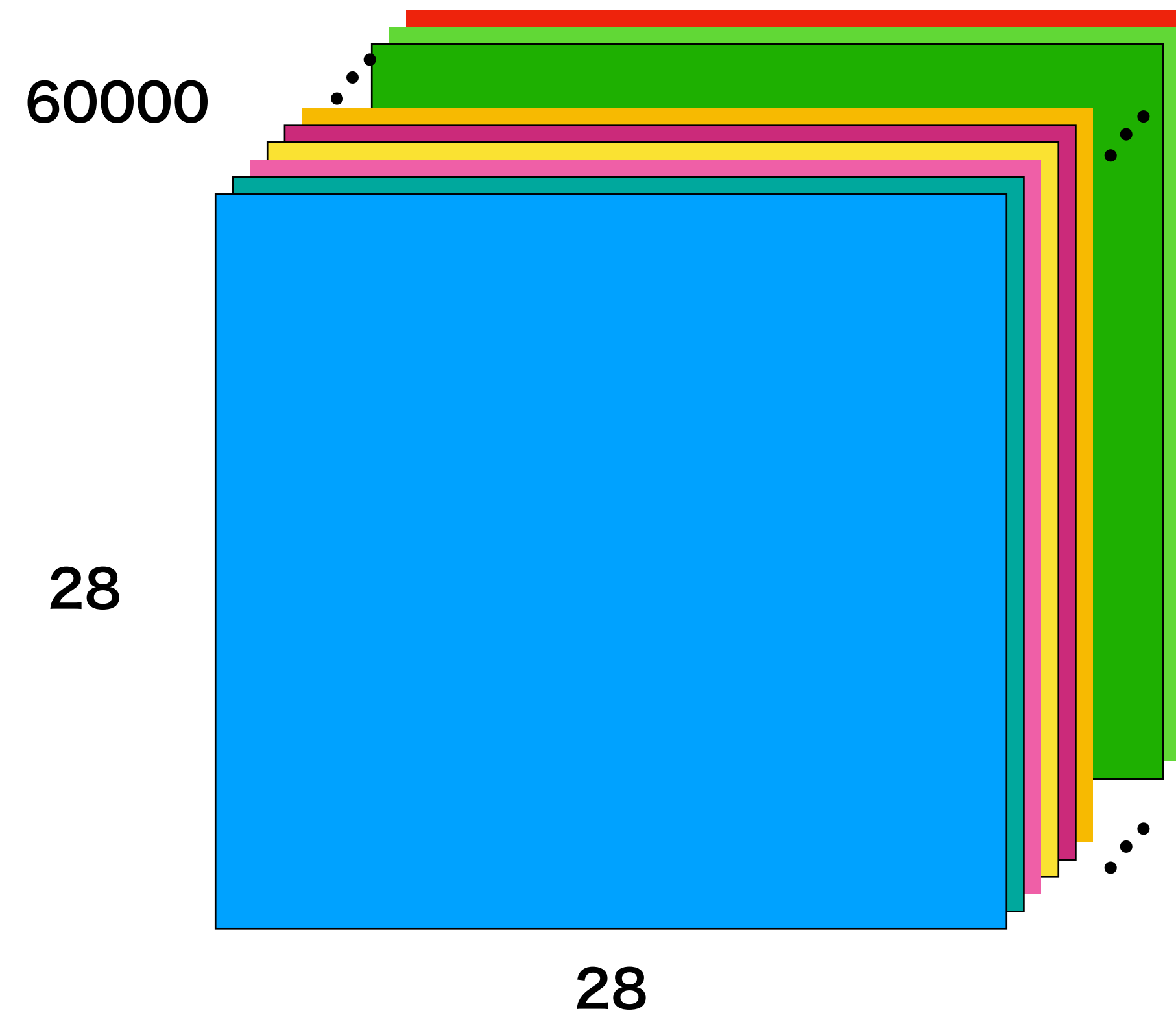
まだ入力しなくていいです

```
x_train = x_train.reshape((x_train.shape[0],784))  
x_test = x_test.reshape((x_test.shape[0],784))  
print(x_train.shape)  
print(x_test.shape)
```

x\_trainのshapeは？

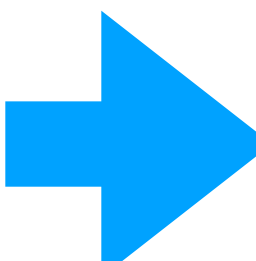
```
print(x_train.shape)  
(60000, 28, 28)
```

x\_train[0]のshapeは？

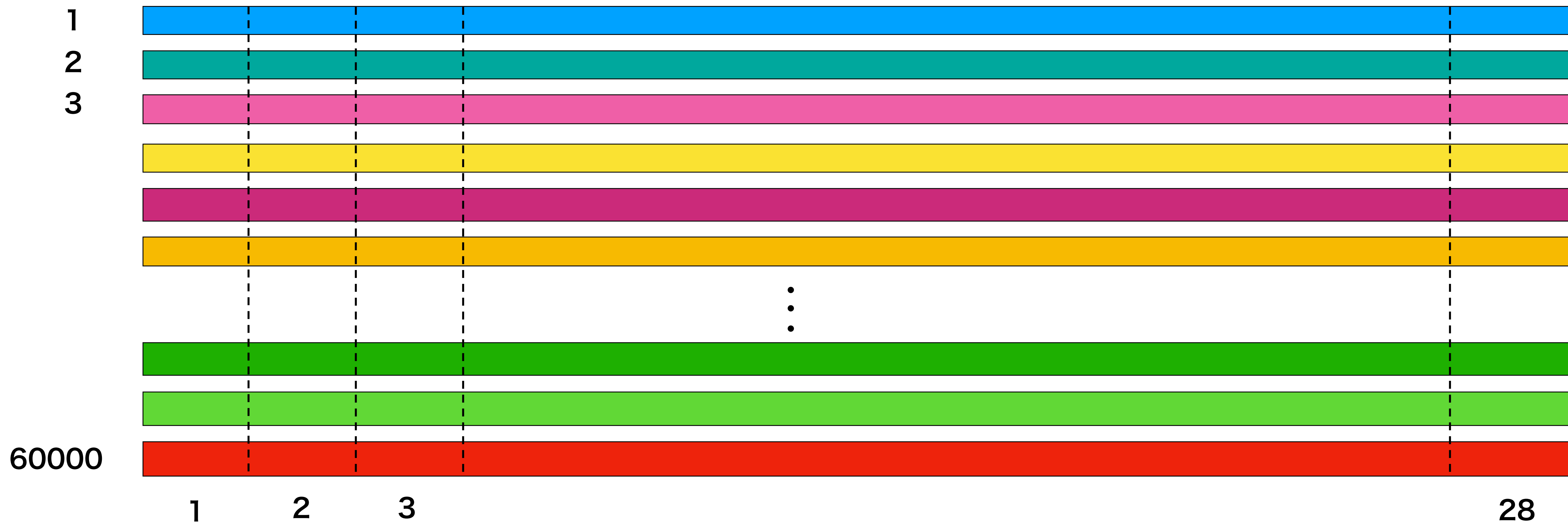


print(x\_train[0].shape)は1枚目の画像の配列なので(28,28)となる

# 画像の2次元配列を1次元配列にしたい

(60000, 28, 28)  (60000, 28 × 28)

$$28 \times 28 = 784$$





# 画像の2次元配列を1次元配列にしたい

reshape()で配列の形状を変える

```
a = np.array([1,2,3,4,5,6,7,8,])  
print(a)  
[1 2 3 4 5 6 7 8]
```

aを(2,4)に変える

```
a = a.reshape(2,4)  
print(a)  
[[1 2 3 4]  
 [5 6 7 8]]
```

# 画像の2次元配列を1次元配列にしたい

reshape()で配列の形状を変える

```
a = np.array([1,2,3,4,5,6,7,8,])  
print(a)  
[1 2 3 4 5 6 7 8]
```

aを(2,4)に変える

```
a = a.reshape(2,4)  
print(a)  
  
[[1 2 3 4]  
 [5 6 7 8]]
```

aを(2,2,2)に変える

```
a = a.reshape(2,2,2)  
print(a)
```

```
[[[1 2]  
  [3 4]  
  [5 6]  
  [ 7 8]]]
```

要素の合計が合っていれば  
どの形にも変えられる

# 画像の2次元配列を1次元配列にしたい

reshape()で配列の形状を変える

```
a = np.array([1,2,3,4,5,6,7,8,])  
print(a)  
[1 2 3 4 5 6 7 8]
```

aを(2,4)に変える

```
a = a.reshape(2,4)  
print(a)  
[[1 2 3 4]  
 [5 6 7 8]]
```

```
print(x_train.shape)  
(60000, 28, 28)
```

→ x\_train.shape[0]  
(60000, 28, 28)の1つ目なので60000

```
x_train = x_train.reshape(60000,784)  
x_test = x_test.reshape(10000,784)
```

```
x_train = x_train.reshape((x_train.shape[0],784))  
x_test = x_test.reshape((x_test.shape[0],784))  
print(x_train.shape)  
print(x_test.shape)
```

```
(60000, 784)  
(10000, 784)
```

## 深層学習前のデータの整理

x\_train (特徴量)

- 正規化する

```
x_train = x_train / 255  
x_test = x_test / 255
```

データを特定の範囲  
や形式に変換する



配列の数字は0~255のいずれか  
全てを255で割って0~1の間に変換する

numpy配列は四則演算が  
それぞれの要素に行なわれます

```
a = a.reshape(2,2,2)  
print(a)  
print(a.shape)  
  
[[[1 2]  
  [3 4]]  
  
 [[5 6]  
  [7 8]]]  
(2, 2, 2)
```



```
a = a / 10  
print(a)  
  
[[[0.1 0.2]  
  [0.3 0.4]]  
  
 [[0.5 0.6]  
  [0.7 0.8]]]
```

(x\_train = x\_train / 255 は省略して x\_train /= 255 と書くことも出来ます)

# 深層学習前のデータの整理

**y\_train (正解)**

- one-hot encoding

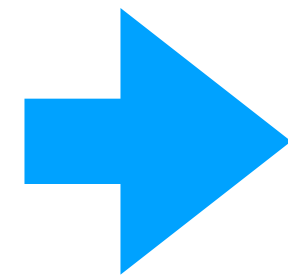
正解は全て0から9のいずれか

これを全て0と1だけで表現するための方法(理由は後述)



# one-hot encoding

0
1
2
3
4
5
6
7
8
9



1, 0, 0, 0, 0, 0, 0, 0, 0, 0 , 0  
0, 1, 0, 0, 0, 0, 0, 0, 0, 0 , 0  
0, 0, 1, 0, 0, 0, 0, 0, 0, 0 , 0  
0, 0, 0, 1, 0, 0, 0, 0, 0, 0 , 0  
0, 0, 0, 0, 1, 0, 0, 0, 0, 0 , 0  
0, 0, 0, 0, 0, 1, 0, 0, 0, 0 , 0  
0, 0, 0, 0, 0, 0, 1, 0, 0, 0 , 0  
0, 0, 0, 0, 0, 0, 0, 1, 0, 0 , 0  
0, 0, 0, 0, 0, 0, 0, 0, 1, 0 , 0  
0, 0, 0, 0, 0, 0, 0, 0, 0, 0 , 1

0と1だけで0から9の数字を表現する

## to\_categorical()関数を使う

```
from keras.utils import to_categorical
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

```
[18] print(y_train.shape)
      print(y_test.shape)
```

```
(60000, 10)
(10000, 10)
```

```
▶ print(y_train[0:10])
```

```
⇒ [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
    [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
    [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
    [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
    [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
    [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
    [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
    [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
    [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
    [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

to\_categorical(変えたい配列, 正解の数)



今回は10クラス

```
print(y_train[0:10])
```

```
[5 0 4 1 9 2 1 3 1 4]
```



# ここまでのまとめ

ノートブックは一度閉じると変数の情報がリセットされるので次回も再度実施します

```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

mnistの読み込み

```
x_train = x_train.reshape(x_train.shape[0], 784)
x_test = x_test.reshape(x_test.shape[0], 784)
print(x_train.shape)
print(x_test.shape)
```

x\_train (特徴量)

- 画像の2次元の配列を1次元にする

```
x_train = x_train / 255
x_test = x_test / 255
```

- 正規化する

```
from keras.utils import to_categorical
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

y\_train (正解)

- one-hot encoding

# 課題

- WebClassにある課題2をやきましょう

締め切りは1週間後の5/2の23:59です。  
締め切りを過ぎた課題は受け取らないので注意して下さい  
(1週間後に正解をアップします)