In [1]:

```
#TensorFlow and tf.kreas
import tensorflow as tf

#Helper libraries
import numpy as np
import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist
#tensorflow 밑에 keras 밑에 datasets 밑에 mnist라는 데이터를 mnist라는 변수에 담는다.
```

In [ ]:

```
각 함수를 불러오는 다른 방법도 있다.
```

In [2]:

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

In [3]:

```
x_train.shape
#이미지는 픽셀. 까만색은 1, 흰색은 0.
#이런 그림이 총 6만 장이 있다.
#그림이기 때문에 28x28.
```

Out[3]:

```
(60000, 28, 28)
```

In [4]:

```
x_train
#거기에 있는 값들을 본다.
```

Out[4]:

```
array([[[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, 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]]], dtype=uint8)
```

In [5]:

```
x_train[0, :, :]
#첫 번째 column의 값을 보자.
```

Out[5]:

```
array([[  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,  30,  36,  94, 154, 170,
        253, 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],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
        253,  70,   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,  35,
        241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  16,  93, 252, 253, 187,   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, 249, 253, 249,  64,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
        148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
```

```
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
        253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
        253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
        195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
         11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   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]], dtype=uint8)
```

In [6]:

```
x_train, x_test = x_train / 255.0, x_test / 255.0
#이런 0~255 사이의 값들을 255.0으로 나눠주면 0부터 1사이의 값으로 수렴된다.
```

In [7]:

```
y_train.shape
#y_train의 shape는 60000개. 그냥 단순한 값들일 것이다. 0이면 0, 1이면 1 등.
#각각의 이미지에 대한 정답 값.
```

Out[7]:

```
(60000,)
```

In [8]:

```
y_train
#각 결과값을 보여준다. 60000개의 그림에 대한 정답 값.
```

Out[8]:

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

In [9]:

```
x_test.shape
#test는 train 할 때 쓰지 않았던 값을 이용해서 이게 잘 돌아가는지 확인해보는 것.
#그림이기 때문에 아까와 마찬가지로 28x28
```

Out[9]:

```
(10000, 28, 28)
```

In [10]:

```
y_test.shape
#x와 y는 개수가 같아야 한다.
```

Out[10]:

```
(10000,)
```

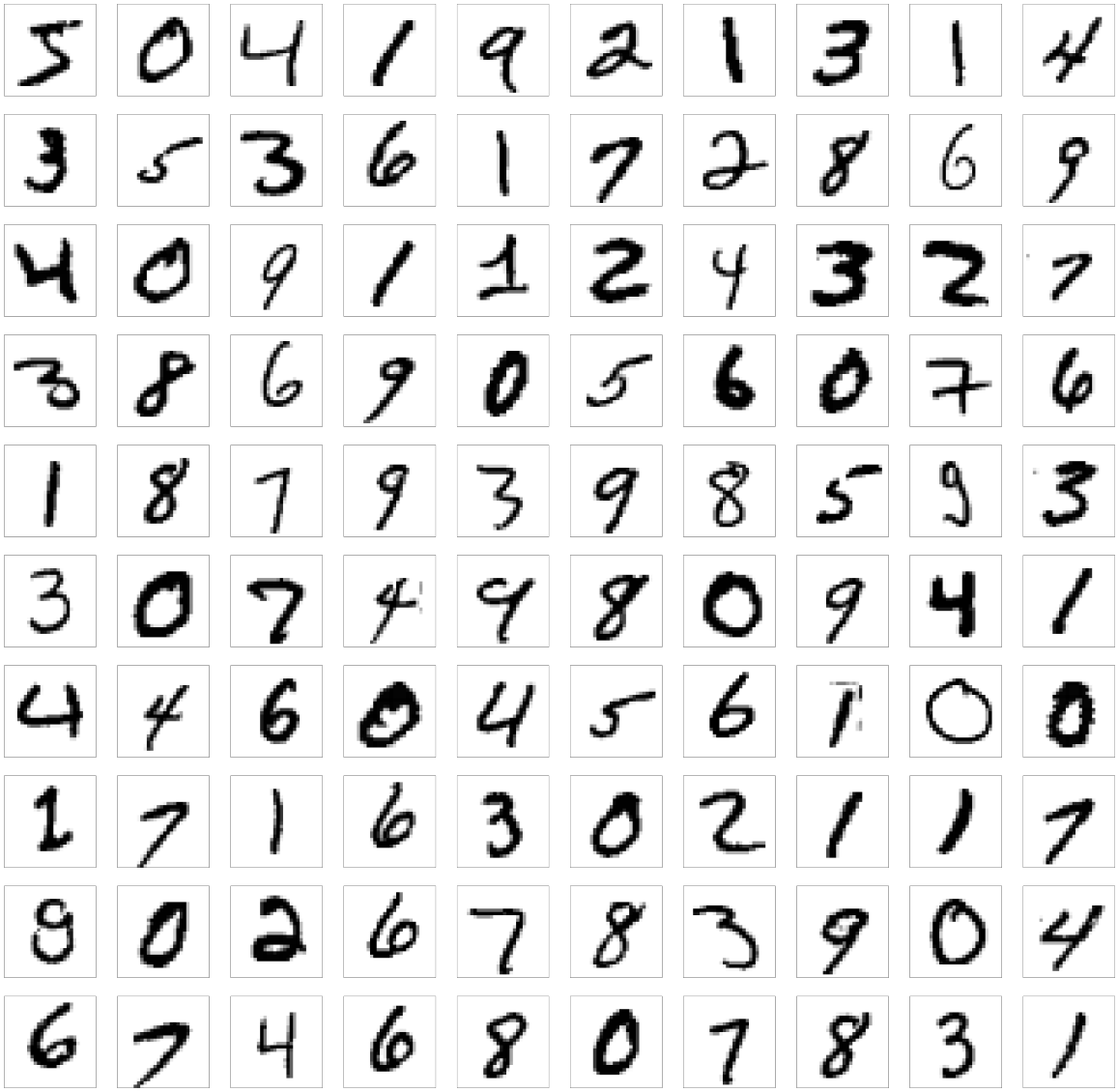
In [11]:

```
plt.figure(figsize = (100,100)) #size of figure
for i in range(100):
    plt.subplot(10, 10, i+1)
    #가로 10개, 세로 10개니까 총 100번. subplot에 index를 붙여주기 위해 i+1을 해준다.

    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_train[i], cmap = plt.cm.binary)
    #imshow(): image를 보여주는 것. x_train을 0부터 99까지.
    #binary: 흑과 백의 binary. 훨씬 더 직관적으로 흑백으로 보여준다.

plt.show()

#한 칸에 가로로 28개, 세로로 28개 칸으로 이루어져 있다.
#28*28 = 784개의 값이 하나의 vector로 만들어진 뒤, 이게 10개의 값으로 나타날텐데,
#10개의 값의 총합은 1로, 다른 것들보다 값이 높게 나온 숫자가 될 것임을 의미.
```



In [12]:

```
img_rows = 28
img_cols = 28
#input이 28x28로 들어가야 한다.
#우리가 보여주는 것은 matrix로, vector로 바꿔주기 위해서는 flatten하는 기능이 필요하다.

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape = (img_rows, img_cols)),
    #여기서 위에서 언급했던 flatten해주는 작업을 한다.
    #이미지같은 matrix 정보가 input으로 들어올 때 이걸 vector 값으로 바꿔주기 위한 기능.

    tf.keras.layers.Dense(512, activation = 'relu'),
    #hidden layer의 activation은 relu로 해주면 좋다.

    #Flatten을 하지 않는다면, Dense(512, input_size(784), actication = 'relu')로 써질 수도 있다.
    #그러나 이렇게 쓰려면 28x28을 하나의 vector로 펴준 다음에 이것을 사용해야 한다.
    #이 단계를 뛰어넘기 위해서 Flatten 작업을 먼저 해준 것.

    tf.keras.layers.Dropout(0),
    #이 줄은 제외해도 잘 넘어간다. performance를 높이는 기능일 뿐.
    #Dropout(0보다 크고 1보다 작은 수)를 넣으면 몇 개를 잘라두는 것이다. 더 잘되기도 한다. 솥치기.

    tf.keras.layers.Dense(10, activation = 'softmax')
])

#Dense가 2개라는 것은 화살표 묶음 세트가 2개라는 것을 의미한다. layer는 총 3개
#input layer = 784개, hidden layer = 512개, output layer = 10개
```

```
WARNING:tensorflow:From C:₩Users₩Hyunseo Choi₩Anaconda3₩lib₩site-packages₩tensorfl
ow₩python₩ops₩resource_variable_ops.py:435: colocate_with (from tensorflow.python.
framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:₩Users₩Hyunseo Choi₩Anaconda3₩lib₩site-packages₩tensorfl
ow₩python₩keras₩layers₩core.py:143: calling dropout (from tensorflow.python.ops.nn
_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_p
rob`.
```

In [13]:

```
x_train.shape
#28x28짜리가 60000 장이 있다.
#(60000, 784)라면 flatten 작업 없이 Dense를 하면 된다.
#x_train.reshape(60000, 784)
#이걸 통해서 shape를 바꿔줄 수 있다. 이 작업을 먼저 거친다면 flatten이 필요가 없다.
```

Out[13]:

```
(60000, 28, 28)
```

In [14]:

```
model.summary()
#이걸 통해 볼 수 있다.
#input layer의 노드의 개수는? 28x28 = 784개.
#첫 번째 hidden layer의 노드의 개수는? 512개.
#output layer의 노드의 개수는? 10개. 0부터 9까지 중 어디에 일치하는지.

#flatten과 dropout은 weight/parameter와는 상관이 없다.
#dense_2의 parameter number = (784 + 1) * 512 = 401920
#dense_3의 parameter number = (512 + 1) * 10 = 5130
#화살표 선들이 다 weight, 즉 훈련되어야 할 숫자값임을 알 수 있다.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

In [15]:

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#classification의 optimizer에서는 무조건 'adam'을 쓴다.
#loss = 'sparse_categorical_crossentropy'
#숫자로 2, 3, 이렇게만 되어 있어도 자동적으로 one hot coding 으로 바꿔준다.
```

In [16]:

```
print(model.input_shape)
print(model.output_shape)
```

```
(None, 28, 28)
(None, 10)
```

In [17]:

```
y_train.shape
#60000x100이 되어야 할 것 같지만 60000만 나온다.
#60000개의 값들이 들어 있는데, y에서는 vector가 필요하다.
#5를 나타내기 위해서는 [0000010000]이 있어야 하고, 0을 나타내기 위해서는 [1000000000]이 있어야
 한다.
#그러나 이 작업을 해주지 않기 위해서 위에서 loss부분을 선언해준 것.

#[1000000000]: one-hot coding
#
```

Out[17]:

```
(60000,)
```