```
import torch
import numpy as np


data = [[1,2],[3,4]]
x = torch.tensor(data)
x
```

```
tensor([[1, 2],
        [3, 4]])
```

```
x.shape
```

```
torch.Size([2, 2])
```

```
x[0]
```

```
tensor([1, 2])
```

```
x.size()
```

```
torch.Size([2, 2])
```

```
np_array = np.array(data)
x2 = torch.from_numpy(np_array)
x2
```

```
tensor([[1, 2],
        [3, 4]])
```

```
x_all_ones = torch.ones_like(x)
print(f"ones Tensor : \n{x_all_ones} \n")
x_rand = torch.rand_like(x,dtype=float)
print(f"random Tensor : \n{x_rand} \n")
```

```
ones Tensor :
tensor([[1, 1],
        [1, 1]])

random Tensor :
tensor([[0.2410, 0.5915],
        [0.5893, 0.2943]], dtype=torch.float64)
```

```
shape = (2,3)
rand_tensor = torch.rand(shape)
ones_tensor = torch.ones(shape)
```

```
zeros_tensor = torch.zeros(shape)
print(rand_tensor)
print(ones_tensor)
print(zeros_tensor)
```

```
tensor([[0.8880, 0.6789, 0.9892],
        [0.0467, 0.1977, 0.8118]])
tensor([[1., 1., 1.],
        [1., 1., 1.]])
tensor([[0., 0., 0.],
        [0., 0., 0.]])
```

```
torch.ones((3,2))
```

```
tensor([[1., 1.],
        [1., 1.],
        [1., 1.]])
```

```
torch.ones((2,3),dtype=int)
```

```
tensor([[1, 1, 1],
        [1, 1, 1]])
```

```
tensor = torch.rand(3,4)
print(f"shape of tensor : {tensor.shape}")
print(f"Datatype of tensor : {tensor.dtype}")
print(f"Device tensor on stored : {tensor.device}")
```

```
shape of tensor : torch.Size([3, 4])
Datatype of tensor : torch.float32
Device tensor on stored : cpu
```

```
device = "cuda" if torch.cuda.is_available else "cpu"
print(f"Using {device} device")
```

```
Using cuda device
```

```
x = x.to(device)
x.device
```

```
device(type='cuda', index=0)
```

```
tensor = torch.ones(4,4)
print("First row : ",tensor[0])
print("Second column : ",tensor[:,1])
print("last column : ",tensor[:,3])
tensor[:,1]=0
print(tensor)
print("Second column : ",tensor[:,1])
```

```
First row :  tensor([1., 1., 1., 1.])
Second column :  tensor([1., 1., 1., 1.])
last column :  tensor([1., 1., 1., 1.])
tensor([[1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.]])
Second column :  tensor([0., 0., 0., 0.])
```

```
t1 = torch.cat([tensor,tensor,tensor],dim=1)
print(t1)
```

```
tensor([[1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.],
        [1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.],
        [1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.],
        [1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.]])
```

```
t2 = torch.cat([tensor,tensor,tensor],dim=1)
print(t2)
```

```
tensor([[1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.],
        [1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.],
        [1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.],
        [1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.]])
```

```
tensor
```

```
tensor([[1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.]])
```

```
tensor.T
```

```
tensor([[1., 1., 1., 1.],
        [0., 0., 0., 0.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]])
```

```
y1 = tensor @ tensor.T
y2 = tensor.matmul(tensor.T)
print(f"{y1}\n{y2}")
```

```
tensor([[3., 3., 3., 3.],
        [3., 3., 3., 3.],
        [3., 3., 3., 3.],
        [3., 3., 3., 3.]])
tensor([[3., 3., 3., 3.],
        [3., 3., 3., 3.],
```

```
               [3., 3., 3., 3.],
               [3., 3., 3., 3.]])
```

```
z1 = tensor * tensor
z2 = tensor.matmul(tensor)
print(f"{z1}\n{z2}")
```

```
     tensor([[1., 0., 1., 1.],
             [1., 0., 1., 1.],
             [1., 0., 1., 1.],
             [1., 0., 1., 1.]])
     tensor([[3., 0., 3., 3.],
             [3., 0., 3., 3.],
             [3., 0., 3., 3.],
             [3., 0., 3., 3.]])
```

```
s = tensor.sum()
s
```

```
     tensor(12.)
```

```
s.item()
```

```
     12.0
```

```
t = torch.ones(5)
print(f"t:{t}")
```

```
     t:tensor([1., 1., 1., 1., 1.])
```

```
n = t.numpy()
print(f"n:{n}")
```

```
     n:[1. 1. 1. 1. 1.]
```

```
t.add_(1)
```

```
     tensor([8., 8., 8., 8., 8.])
```

```
n
```

```
     array([9., 9., 9., 9., 9.], dtype=float32)
```

```
np.add(n,1,out=n)
print(f"t: {t}")
print(f"n: {n}")
```

```
     t: tensor([10., 10., 10., 10., 10.])
     n: [10. 10. 10. 10. 10.]
```

```
import torch
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt


training_data = datasets.MNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to data/MNIST/ra
                                                    9913344/? [00:00<00:00, 6614125.82it/s]
Extracting data/MNIST/raw/train-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to data/MNIST/ra
                                                    29696/? [00:00<00:00, 789370.94it/s]
Extracting data/MNIST/raw/train-labels-idx1-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to data/MNIST/raw
                                                    1649664/? [00:00<00:00, 18823431.69it/s]
Extracting data/MNIST/raw/t10k-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to data/MNIST/raw
                                                    5120/? [00:00<00:00, 153038.61it/s]
Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST/raw
```

```
""
```

✓ 1s completed at 11:20 PM