In [6]:
```
pip install torch
```

```
Collecting torch
  Downloading torch-2.2.2-cp311-none-macosx_10_9_x86_64.whl.metadata (25 kB)
Collecting filelock (from torch)
  Downloading filelock-3.16.0-py3-none-any.whl.metadata (3.0 kB)
Requirement already satisfied: typing-extensions>=4.8.0 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from torch) (4.12.2)
Collecting sympy (from torch)
  Downloading sympy-1.13.2-py3-none-any.whl.metadata (12 kB)
Collecting networkx (from torch)
  Downloading networkx-3.3-py3-none-any.whl.metadata (5.1 kB)
Requirement already satisfied: jinja2 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from torch) (3.1.4)
Collecting fsspec (from torch)
  Downloading fsspec-2024.9.0-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: MarkupSafe>=2.0 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from jinja2->torch) (2.1.5)
Collecting mpmath<1.4,>=1.1.0 (from sympy->torch)
  Downloading mpmath-1.3.0-py3-none-any.whl.metadata (8.6 kB)
Downloading torch-2.2.2-cp311-none-macosx_10_9_x86_64.whl (150.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 150.8/150.8 MB 719.0 kB/s eta 0:00:0000:0100:06
Downloading filelock-3.16.0-py3-none-any.whl (16 kB)
Downloading fsspec-2024.9.0-py3-none-any.whl (179 kB)
Downloading networkx-3.3-py3-none-any.whl (1.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 576.0 kB/s eta 0:00:00a 0:00:01
Downloading sympy-1.13.2-py3-none-any.whl (6.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.2/6.2 MB 675.4 kB/s eta 0:00:00a 0:00:01
Downloading mpmath-1.3.0-py3-none-any.whl (536 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 536.2/536.2 kB 802.7 kB/s eta 0:00:00:--:--
Installing collected packages: mpmath, sympy, networkx, fsspec, filelock, torch
Successfully installed filelock-3.16.0 fsspec-2024.9.0 mpmath-1.3.0 networkx-3.3 sympy-1.13.2 torch-2.2.2
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
import torch
x = torch.arange(12, dtype=torch.float32)
x
```

Out[4]:
```
tensor([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.])
```

In [5]:
```
x.numel()
```

Out[5]:
```
12
```

In [7]:
```
x.shape
```

Out[7]:
```
torch.Size([12])
```

In [8]:
```
X = x.reshape(3, 4) #어차피 나눠지니까 둘 중 하나만 알면 나머지에 -1넣으면됨
X
```

Out[8]:
```
tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.]])
```

In [9]:
```
torch.zeros((2, 3, 4))
```

Out[9]:
```
tensor([[[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]],

        [[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]]])
```

In [10]:
```
torch.ones((2, 3, 4))
```

Out[10]:
```
tensor([[[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]],

        [[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]]])
```

In [11]:
```
torch.randn(3, 4)
```

Out[11]:
```
tensor([[ 0.0169,  0.0614,  0.1916, -2.0591],
        [-0.5592, -0.2677,  0.8810,  0.0313],
        [-1.1016, -1.3585, -0.9030,  0.2280]])
```

In [12]:
```
torch.tensor([[2, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
```

Out[12]:
```
tensor([[2, 1, 4, 3],
        [1, 2, 3, 4],
        [4, 3, 2, 1]])
```

In [13]:
```
X[-1], X[1:3]
```

Out[13]:
```
(tensor([ 8.,  9., 10., 11.]),
 tensor([[ 4.,  5.,  6.,  7.],
         [ 8.,  9., 10., 11.]]))
```

In [14]:
```
X[1, 2] = 17
X
```

Out[14]:
```
tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5., 17.,  7.],
        [ 8.,  9., 10., 11.]])
```

In [15]:
```
X[:2, :] = 12
X
```

Out[15]:
```
tensor([[12., 12., 12., 12.],
        [12., 12., 12., 12.],
        [ 8.,  9., 10., 11.]])
```

In [16]:
```
torch.exp(x)
```

Out[16]:
```
tensor([162754.7969, 162754.7969, 162754.7969, 162754.7969, 162754.7969,
        162754.7969, 162754.7969, 162754.7969,   2980.9580,   8103.0840,
         22026.4648,  59874.1406])
```

In [17]:
```
x = torch.tensor([1.0, 2, 4, 8])
y = torch.tensor([2, 2, 2, 2])
x + y, x - y, x * y, x / y, x ** y
```

Out[17]:
```
(tensor([ 3.,  4.,  6., 10.]),
 tensor([-1.,  0.,  2.,  6.]),
 tensor([ 2.,  4.,  8., 16.]),
 tensor([0.5000, 1.0000, 2.0000, 4.0000]),
 tensor([ 1.,  4., 16., 64.]))
```

In [18]:
```
X = torch.arange(12, dtype=torch.float32).reshape((3,4))
Y = torch.tensor([[2.0, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
torch.cat((X, Y), dim=0), torch.cat((X, Y), dim=1)
```

Out[18]:
```
(tensor([[ 0.,  1.,  2.,  3.],
         [ 4.,  5.,  6.,  7.],
         [ 8.,  9., 10., 11.],
         [ 2.,  1.,  4.,  3.],
         [ 1.,  2.,  3.,  4.],
         [ 4.,  3.,  2.,  1.]]),
 tensor([[ 0.,  1.,  2.,  3.,  2.,  1.,  4.,  3.],
         [ 4.,  5.,  6.,  7.,  1.,  2.,  3.,  4.],
         [ 8.,  9., 10., 11.,  4.,  3.,  2.,  1.]]))
```

In [19]:
```
X == Y
```

Out[19]:
```
tensor([[False,  True, False,  True],
        [False, False, False, False],
        [False, False, False, False]])
```

In [20]:
```
X.sum()
```

Out[20]:
```
tensor(66.)
```

In [21]:
```
a = torch.arange(3).reshape((3, 1))
b = torch.arange(2).reshape((1, 2))
a, b
```

Out[21]:
```
(tensor([[0],
         [1],
         [2]]),
 tensor([[0, 1]]))
```

In [22]:
```
a + b
```

Out[22]:
```
tensor([[0, 1],
        [1, 2],
        [2, 3]])
```

In [23]:
```
before = id(Y)
Y = Y + X
```

```
id(Y) == before
```

Out[23]: False

In [24]:
```
Z = torch.zeros_like(Y)
print('id(Z):', id(Z))
Z[:] = X + Y
print('id(Z):', id(Z))
```

```
id(Z): 4698319152
id(Z): 4698319152
```

In [25]:
```
before = id(X)
X += Y
id(X) == before
```

Out[25]: True

In [26]:
```
A = X.numpy()
B = torch.from_numpy(A)
type(A), type(B)
```

Out[26]: (numpy.ndarray, torch.Tensor)

In [27]:
```
a = torch.tensor([3.5])
a, a.item(), float(a), int(a)
```

Out[27]: (tensor([3.5000]), 3.5, 3.5, 3)

Discussion and Takeaway message: In this section, we learned about basic grammar and functions that deal with arrangements with pytorch. What I learned newly was that when rescuing, even if only one of the two numbers was written and the rest of the numbers were filled in with -1, the code still performed well. In addition, I learned a new function 'torch.cat ()' that connects two or more tensors according to a given dimension. It was a unit that also provided tips for managing memory efficiently.