In [1]:
```
import torch
```

In [2]:
```
x = torch.arange(12, dtype=torch.float32)
x
```

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

In [3]:
```
x.numel()
```

Out[3]:
```
12
```

In [4]:
```
x.shape
```

Out[4]:
```
torch.Size([12])
```

In [5]:
```
X = x.reshape(3, 4)
X
```

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

In [6]:
```
torch.zeros((2, 3, 4))
```

Out[6]:
```
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 [7]:
```
torch.ones((2, 3, 4))
```

Out[7]:
```
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 [8]:
```
torch.randn(3, 4)
```

Out[8]:
```
tensor([[ 0.2484,  0.4358, -0.1332, -1.3240],
        [ 0.9454,  0.8692, -1.3550, -1.4533],
        [-0.3344, -1.2855, -0.9603,  1.8054]])
```

In [9]:
```
torch.tensor([[2, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
```

Out[9]:
```
tensor([[2, 1, 4, 3],
        [1, 2, 3, 4],
        [4, 3, 2, 1]])
```

In [10]:
```
X[-1], X[1:3]
```

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

In [11]:
```
X[1, 2] = 17
X
```

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

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

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

In [13]:
```
torch.exp(x)
```

Out[13]:
```
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 [14]:
```
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[14]:
```
(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 [15]:
```
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[15]:
```
(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 [16]:
```
X == Y
```

Out[16]:
```
tensor([[False,  True, False,  True],
        [False, False, False, False],
        [False, False, False, False]])
```

In [17]:
```
X.sum()
```

Out[17]:
```
tensor(66.)
```

In [18]:
```
a = torch.arange(3).reshape((3, 1))
b = torch.arange(2).reshape((1, 2))
a, b
```

Out[18]:
```
(tensor([[0],
         [1],
         [2]]),
 tensor([[0, 1]]))
```

In [19]:
```
a + b
```

Out[19]:
```
tensor([[0, 1],
        [1, 2],
        [2, 3]])
```

In [20]:
```
before = id(Y)
Y = Y + X
id(Y) == before
```

Out[20]:
```
False
```

In [21]:
```
Z = torch.zeros_like(Y)
print('id(Z):', id(Z))
Z[:] = X + Y
print('id(Z):', id(Z))
```

```
id(Z): 4575857072
id(Z): 4575857072
```

In [22]:
```
before = id(X)
X += Y
id(X) == before
```

Out[22]:
```
True
```

In [23]:
```
A = X.numpy()
B = torch.from_numpy(A)
type(A), type(B)
```

Out[23]:
```
(numpy.ndarray, torch.Tensor)
```

In [24]:
```
a = torch.tensor([3.5])
a, a.item(), float(a), int(a)
```

Out[24]:
```
(tensor([3.5000]), 3.5, 3.5, 3)
```

In [ ]:
```
```

Section 2.3

In [25]:
```
x = torch.tensor(3.0)
y = torch.tensor(2.0)

x + y, x * y, x / y, x**y
```

Out[25]:
```
(tensor(5.), tensor(6.), tensor(1.5000), tensor(9.))
```

In [26]:
```
x = torch.arange(3)
x
```

Out[26]:
```
tensor([0, 1, 2])
```

In [27]:
```
x[2]
```

Out[27]:
```
tensor(2)
```

In [28]:
```
len(x)
```

Out[28]:
```
3
```

In [29]:
```
x.shape
```

Out[29]:
```
torch.Size([3])
```

In [30]:
```
A = torch.arange(6).reshape(3, 2)
A
```

Out[30]:
```
tensor([[0, 1],
        [2, 3],
        [4, 5]])
```

In [31]:
```
A.T
```

Out[31]:
```
tensor([[0, 2, 4],
        [1, 3, 5]])
```

In [32]:
```
A = torch.tensor([[1, 2, 3], [2, 0, 4], [3, 4, 5]])
A == A.T
```

Out[32]:
```
tensor([[True, True, True],
        [True, True, True],
        [True, True, True]])
```

In [33]:
```
torch.arange(24).reshape(2, 3, 4)
```

Out[33]:
```
tensor([[[ 0,  1,  2,  3],
         [ 4,  5,  6,  7],
         [ 8,  9, 10, 11]],

        [[12, 13, 14, 15],
         [16, 17, 18, 19],
         [20, 21, 22, 23]]])
```

In [34]:
```
A = torch.arange(6, dtype=torch.float32).reshape(2, 3)
B = A.clone()  # Assign a copy of A to B by allocating new memory
```

```
A, A + B
```

Out[34]:
```
(tensor([[0., 1., 2.],
         [3., 4., 5.]]),
 tensor([[ 0.,  2.,  4.],
         [ 6.,  8., 10.]]))
```

In [35]:
```
A * B
```

Out[35]:
```
tensor([[ 0.,  1.,  4.],
        [ 9., 16., 25.]])
```

In [36]:
```
a = 2
X = torch.arange(24).reshape(2, 3, 4)
a + X, (a * X).shape
```

Out[36]:
```
(tensor([[[ 2,  3,  4,  5],
          [ 6,  7,  8,  9],
          [10, 11, 12, 13]],

         [[14, 15, 16, 17],
          [18, 19, 20, 21],
          [22, 23, 24, 25]]]),
 torch.Size([2, 3, 4]))
```

In [37]:
```
x = torch.arange(3, dtype=torch.float32)
x, x.sum()
```

Out[37]:
```
(tensor([0., 1., 2.]), tensor(3.))
```

In [38]:
```
A.shape, A.sum()
```

Out[38]:
```
(torch.Size([2, 3]), tensor(15.))
```

In [39]:
```
A.shape, A.sum(axis=0).shape
```

Out[39]:
```
(torch.Size([2, 3]), torch.Size([3]))
```

In [40]:
```
A.shape, A.sum(axis=1).shape
```

Out[40]:
```
(torch.Size([2, 3]), torch.Size([2]))
```

In [41]:
```
A.shape, A.sum(axis=1).shape
```

Out[41]:
```
(torch.Size([2, 3]), torch.Size([2]))
```

In [42]:
```
A.mean(), A.sum() / A.numel()
```

Out[42]:
```
(tensor(2.5000), tensor(2.5000))
```

In [43]:
```
A.mean(axis=0), A.sum(axis=0) / A.shape[0]
```

Out[43]:
```
(tensor([1.5000, 2.5000, 3.5000]), tensor([1.5000, 2.5000, 3.5000]))
```

In [44]:
```
sum_A = A.sum(axis=1, keepdims=True)
sum_A, sum_A.shape
```

Out[44]:
```
(tensor([[ 3.],
         [12.]]),
 torch.Size([2, 1]))
```

In [45]:
```
A / sum_A
```

Out[45]:
```
tensor([[0.0000, 0.3333, 0.6667],
        [0.2500, 0.3333, 0.4167]])
```

In [46]:
```
A.cumsum(axis=0)
```

Out[46]:
```
tensor([[0., 1., 2.],
        [3., 5., 7.]])
```

In [47]:
```
y = torch.ones(3, dtype = torch.float32)
x, y, torch.dot(x, y)
```

Out[47]:
```
(tensor([0., 1., 2.]), tensor([1., 1., 1.]), tensor(3.))
```

In [48]:
```
torch.sum(x * y)
```

Out[48]:
```
tensor(3.)
```

In [49]:
```
A.shape, x.shape, torch.mv(A, x), A@x
```

Out[49]:
```
(torch.Size([2, 3]), torch.Size([3]), tensor([ 5., 14.]), tensor([ 5., 1
4.]))
```

In [50]:
```
B = torch.ones(3, 4)
torch.mm(A, B), A@B
```

Out[50]:
```
(tensor([[ 3.,  3.,  3.,  3.],
         [12., 12., 12., 12.]]),
 tensor([[ 3.,  3.,  3.,  3.],
         [12., 12., 12., 12.]]))
```

In [51]:
```
u = torch.tensor([3.0, -4.0])
torch.norm(u)
```

Out[51]:
```
tensor(5.)
```

In [52]:
```
torch.norm(torch.ones((4, 9)))
```

Out[52]:
```
tensor(6.)
```

Problem 2

In [55]:
```
# User 1 preferences
q1 = [2, 0, 1]
q2 = [1, 2, 0]
q3 = [0, 1, 3]
```

```
# Movie 1 attributes
k1 = [1, 0, 2]
k2 = [0, 1, 1]
k3 = [2, 1, 0]
k4 = [1, 2, 1]
```

Question A

In [57]:
```
Q = torch.tensor([q1, q2, q3])
K = torch.tensor([k1, k2, k3, k4])
Q, K
```

Out[57]:
```
(tensor([[2, 0, 1],
         [1, 2, 0],
         [0, 1, 3]]),
 tensor([[1, 0, 2],
         [0, 1, 1],
         [2, 1, 0],
         [1, 2, 1]]))
```

Question B

In [65]:
```
Q
```

Out[65]:
```
tensor([[2, 0, 1],
        [1, 2, 0],
        [0, 1, 3]])
```

In [64]:
```
K.T
```

Out[64]:
```
tensor([[1, 0, 2, 1],
        [0, 1, 1, 2],
        [2, 1, 0, 1]])
```

In [63]:
```
S = torch.matmul(Q, K.T)
S
```

Out[63]:
```
tensor([[4, 1, 4, 3],
        [1, 2, 4, 5],
        [6, 4, 1, 5]])
```

Question C

The row repersent the users score for Q_rownum and the column represents the score for the movie attribute k_colnum

so row 1 column 1 is the score for user q1 and movie k1

row 2 and column 1 is the score for user q2 and movie k1

etc....

Question D

In [66]:
```
S
```

Out[66]:
```
tensor([[4, 1, 4, 3],
        [1, 2, 4, 5],
        [6, 4, 1, 5]])
```

In [ ]: