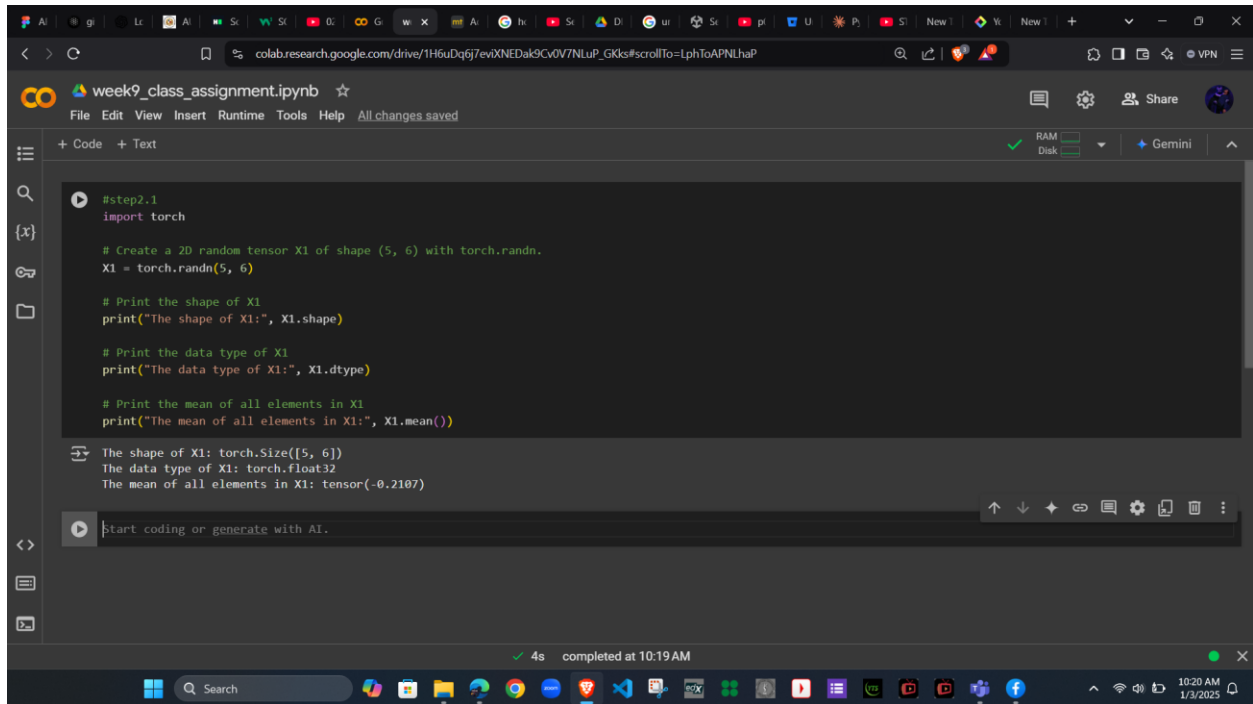


# Part 1: Basic Pytorch

## Step 2.1 :



The screenshot shows a Google Colab notebook titled "week9\_class\_assignment.ipynb". The code cell contains the following Python code:

```
#step2.1
import torch

# Create a 2D random tensor X1 of shape (5, 6) with torch.randn.
X1 = torch.randn(5, 6)

# Print the shape of X1
print("The shape of X1:", X1.shape)

# Print the data type of X1
print("The data type of X1:", X1.dtype)

# Print the mean of all elements in X1
print("The mean of all elements in X1:", X1.mean())
```

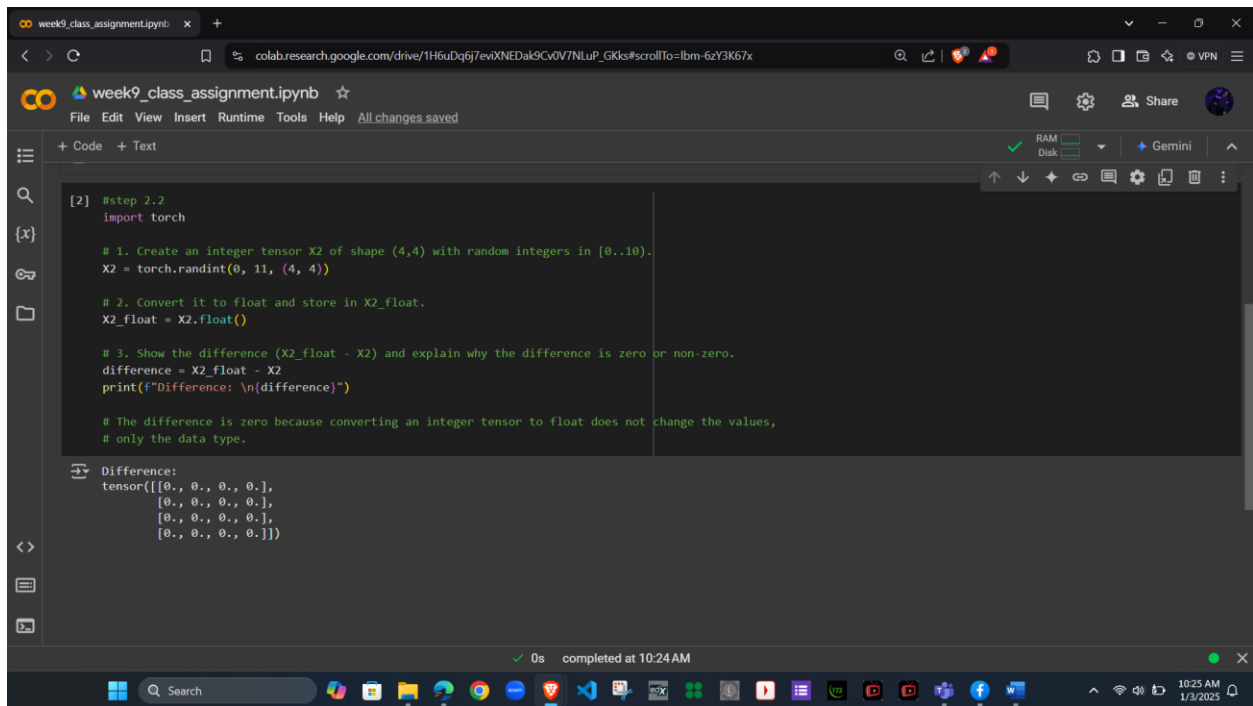
The output of the code cell is:

```
The shape of X1: torch.Size([5, 6])
The data type of X1: torch.float32
The mean of all elements in X1: tensor(-0.2107)
```

Below the code cell, there is a text input field with the placeholder text "Start coding or generate with AI." and a button to start coding.

The bottom status bar indicates that the code was completed at 10:19 AM.

## Step 2.2:



The screenshot shows a Google Colab notebook titled "week9\_class\_assignment.ipynb". The code in the notebook is as follows:

```
[2] #step 2.2
import torch

# 1. Create an integer tensor X2 of shape (4,4) with random integers in [0..10].
X2 = torch.randint(0, 11, (4, 4))

# 2. Convert it to float and store in X2_float.
X2_float = X2.float()

# 3. Show the difference (X2_float - X2) and explain why the difference is zero or non-zero.
difference = X2_float - X2
print(f"Difference: \n{difference}")

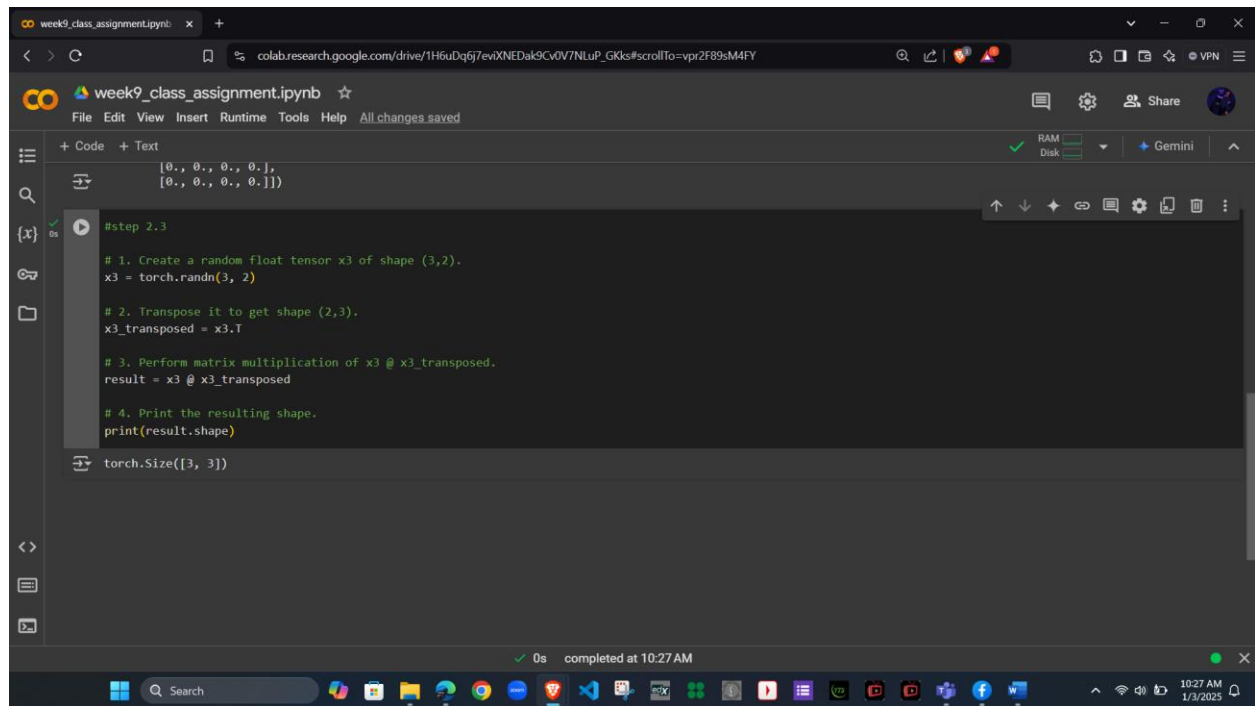
# The difference is zero because converting an integer tensor to float does not change the values,
# only the data type.
```

The output of the code is:

```
Difference:
tensor([[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]])
```

The bottom status bar indicates the code was completed at 10:24 AM.

## Step 2.3:

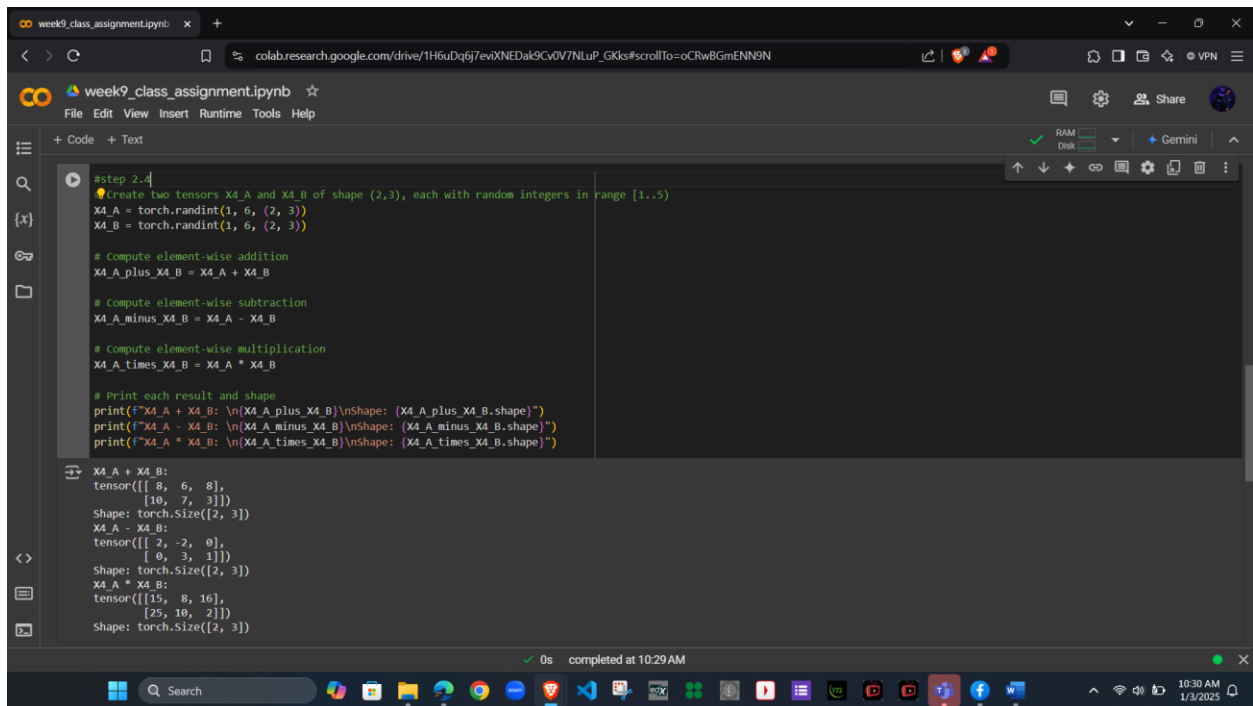


The screenshot shows a Google Colab notebook titled "week9\_class\_assignment.ipynb". The code in the notebook is as follows:

```
[0., 0., 0., 0.],  
[0., 0., 0., 0.]]  
  
# Step 2.3  
  
# 1. Create a random float tensor x3 of shape (3,2).  
x3 = torch.randn(3, 2)  
  
# 2. Transpose it to get shape (2,3).  
x3_transposed = x3.T  
  
# 3. Perform matrix multiplication of x3 @ x3_transposed.  
result = x3 @ x3_transposed  
  
# 4. Print the resulting shape.  
print(result.shape)  
  
torch.Size([3, 3])
```

The notebook interface shows the code is executed successfully, with a status bar indicating "0s completed at 10:27 AM". The Windows taskbar at the bottom shows the time as 10:27 AM on 1/3/2023.

## Step 2.4:



```
#step 2.4
# Create two tensors X4_A and X4_B of shape (2,3), each with random integers in range [1..5]
X4_A = torch.randint(1, 6, (2, 3))
X4_B = torch.randint(1, 6, (2, 3))

# Compute element-wise addition
X4_A_plus_X4_B = X4_A + X4_B

# Compute element-wise subtraction
X4_A_minus_X4_B = X4_A - X4_B

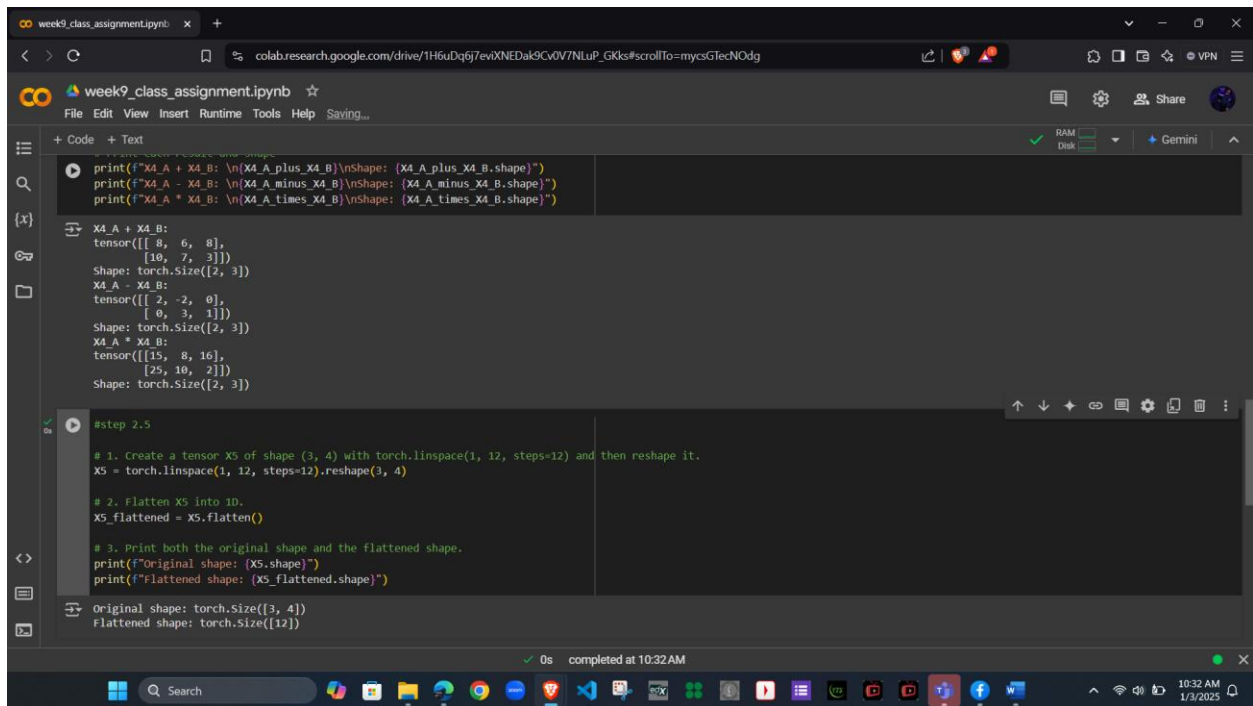
# Compute element-wise multiplication
X4_A_times_X4_B = X4_A * X4_B

# Print each result and shape
print(f"X4_A + X4_B: \n{X4_A_plus_X4_B}\nShape: {X4_A_plus_X4_B.shape}")
print(f"X4_A - X4_B: \n{X4_A_minus_X4_B}\nShape: {X4_A_minus_X4_B.shape}")
print(f"X4_A * X4_B: \n{X4_A_times_X4_B}\nShape: {X4_A_times_X4_B.shape}")
```

X4\_A + X4\_B:  
tensor([[ 8, 6, 8],  
 [10, 7, 3]])  
Shape: torch.Size([2, 3])  
X4\_A - X4\_B:  
tensor([[ 2, -2, 0],  
 [ 0, 3, 1]])  
Shape: torch.Size([2, 3])  
X4\_A \* X4\_B:  
tensor([[15, 8, 16],  
 [25, 10, 2]])  
Shape: torch.Size([2, 3])

0s completed at 10:29 AM

## Step 2.5:



The screenshot shows a Google Colab notebook titled "week9\_class\_assignment.ipynb". The top toolbar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", "Help", and "Saving...". The left sidebar contains icons for file explorer, search, and other notebook functions. The main area displays two code cells. The first cell contains code for tensor operations: addition, subtraction, and multiplication of two tensors, X4\_A and X4\_B, with their respective shapes printed. The second cell, titled "#step 2.5", contains code for creating a tensor X5, flattening it, and printing the original and flattened shapes. The output of the second cell shows the original shape as torch.Size([3, 4]) and the flattened shape as torch.Size([12]). The bottom status bar indicates the notebook is "completed at 10:32 AM".

```
print("X4_A + X4_B: \n(X4_A plus X4_B)\nShape: {X4_A_plus_X4_B.shape}")
print("X4_A - X4_B: \n(X4_A minus X4_B)\nShape: {X4_A_minus_X4_B.shape}")
print("X4_A * X4_B: \n(X4_A times X4_B)\nShape: {X4_A_times_X4_B.shape}")

X4_A + X4_B:
tensor([[ 8,  6,  8],
        [10,  7,  3]])
Shape: torch.Size([2, 3])
X4_A - X4_B:
tensor([[ 2, -2,  0],
        [ 0,  3,  1]])
Shape: torch.Size([2, 3])
X4_A * X4_B:
tensor([[15,  8, 16],
        [25, 10,  2]])
Shape: torch.Size([2, 3])

#step 2.5

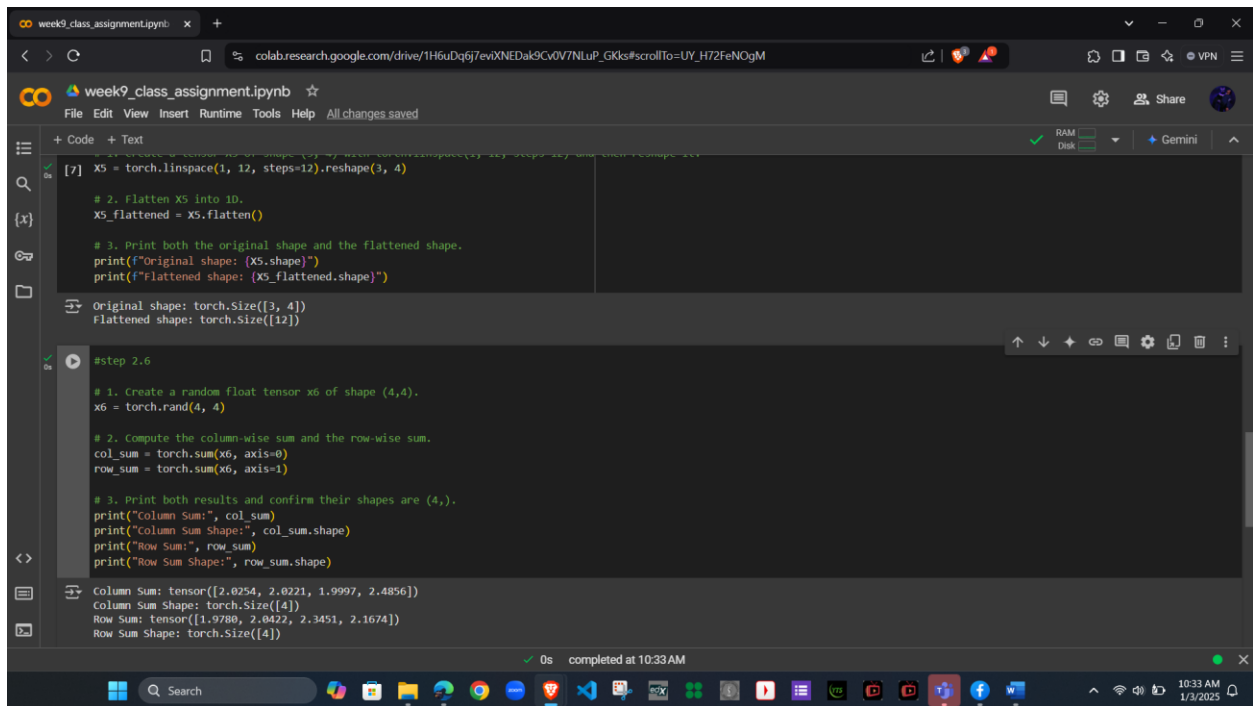
# 1. Create a tensor X5 of shape (3, 4) with torch.linspace(1, 12, steps=12) and then reshape it.
X5 = torch.linspace(1, 12, steps=12).reshape(3, 4)

# 2. Flatten X5 into 1D.
X5_flattened = X5.flatten()

# 3. Print both the original shape and the flattened shape.
print("Original shape: {X5.shape}")
print("Flattened shape: {X5_flattened.shape}")

Original shape: torch.Size([3, 4])
Flattened shape: torch.Size([12])
```

## Step 2.6:



```
week9_class_assignment.ipynb
colab.research.google.com/drive/1H6uDq6j7ewiXNEDak9CvOV7NLuP_GKks#scrollTo=UY_H72FeNogM

week9_class_assignment.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text
[7] X5 = torch.linspace(1, 12, steps=12).reshape(3, 4)

# 2. Flatten X5 into 1D.
X5_flattened = X5.flatten()

# 3. Print both the original shape and the flattened shape.
print(f"Original shape: {X5.shape}")
print(f"Flattened shape: {X5_flattened.shape}")

Original shape: torch.Size([3, 4])
Flattened shape: torch.Size([12])

#step 2.6

# 1. Create a random float tensor x6 of shape (4,4).
x6 = torch.rand(4, 4)

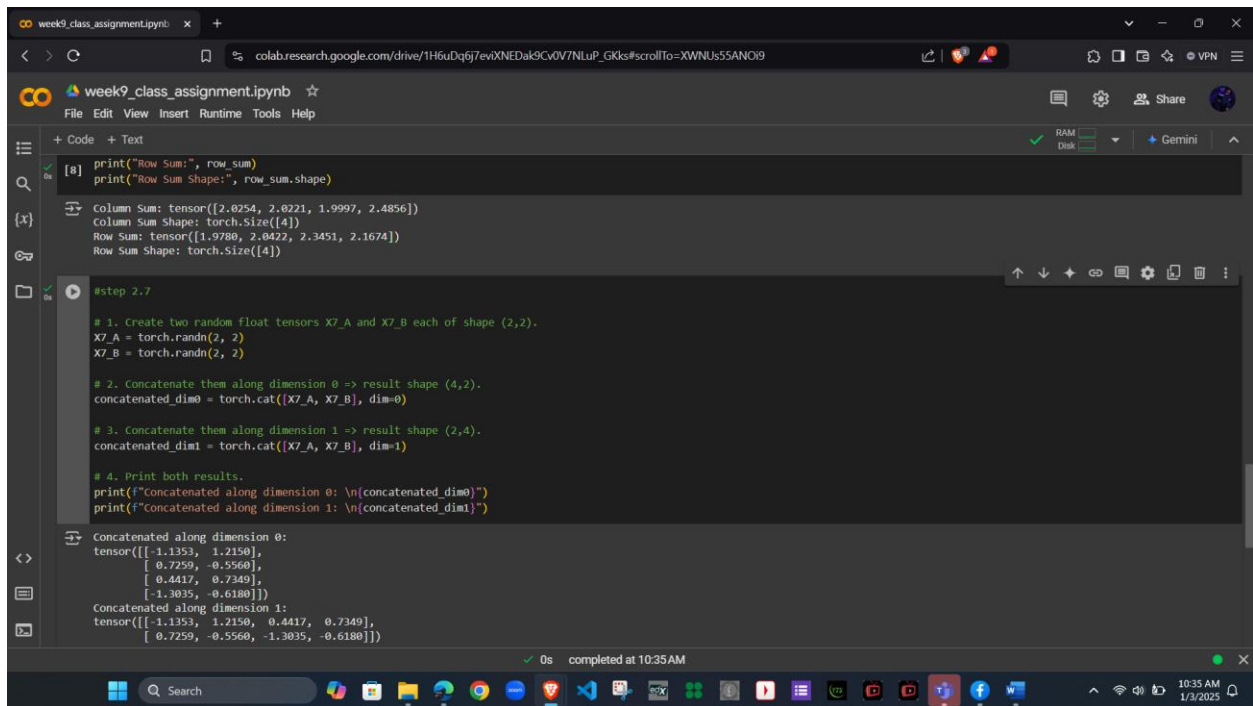
# 2. Compute the column-wise sum and the row-wise sum.
col_sum = torch.sum(x6, axis=0)
row_sum = torch.sum(x6, axis=1)

# 3. Print both results and confirm their shapes are (4,).
print("Column Sum:", col_sum)
print("Column Sum Shape:", col_sum.shape)
print("Row Sum:", row_sum)
print("Row Sum Shape:", row_sum.shape)

Column Sum: tensor([2.0254, 2.0221, 1.9997, 2.4856])
Column Sum Shape: torch.Size([4])
Row Sum: tensor([1.9780, 2.0422, 2.3451, 2.1674])
Row Sum Shape: torch.Size([4])

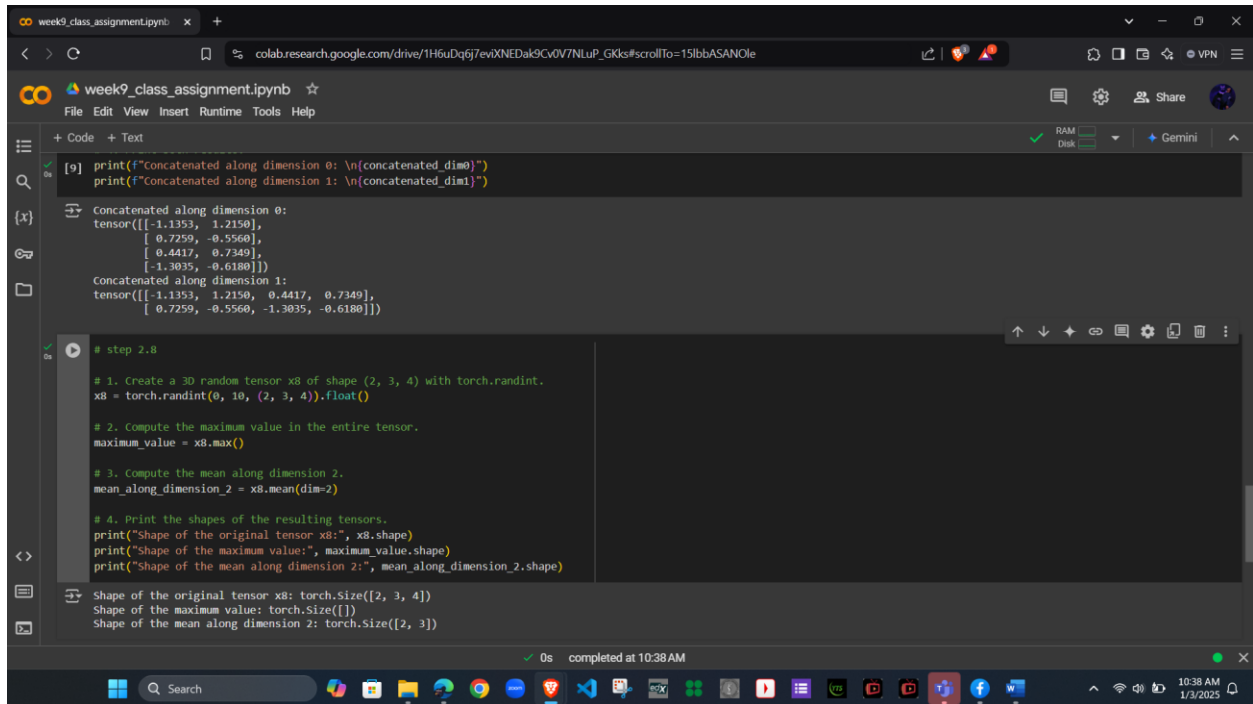
0s completed at 10:33 AM
10:33 AM 1/3/2025
```

## Step 2.7:



```
week9_class_assignment.ipynb x +
colab.research.google.com/drive/1H6uDq6j7ewXNEDak9Cv0V7NluP_GKks#scrollTo=XWNUs55SANOi9
File Edit View Insert Runtime Tools Help
+ Code + Text
[0] print("Row Sum:", row_sum)
    print("Row Sum Shape:", row_sum.shape)
Column Sum: tensor([2.0254, 2.0221, 1.9997, 2.4856])
Column Sum Shape: torch.Size([4])
Row Sum: tensor([1.9780, 2.0422, 2.3451, 2.1674])
Row Sum Shape: torch.Size([4])
#step 2.7
# 1. Create two random float tensors X7_A and X7_B each of shape (2,2).
X7_A = torch.randn(2, 2)
X7_B = torch.randn(2, 2)
# 2. Concatenate them along dimension 0 => result shape (4,2).
concatenated_dim0 = torch.cat([X7_A, X7_B], dim=0)
# 3. Concatenate them along dimension 1 => result shape (2,4).
concatenated_dim1 = torch.cat([X7_A, X7_B], dim=1)
# 4. Print both results.
print("Concatenated along dimension 0: \n(concatenated_dim0)")
print("Concatenated along dimension 1: \n(concatenated_dim1)")
Concatenated along dimension 0:
tensor([[-1.1353,  1.2150],
        [ 0.7259, -0.5560],
        [ 0.4417,  0.7349],
        [-1.3035, -0.6180]])
Concatenated along dimension 1:
tensor([[-1.1353,  1.2150,  0.4417,  0.7349],
        [ 0.7259, -0.5560, -1.3035, -0.6180]])
0s completed at 10:35AM
10:35 AM 1/3/2025
```

## Step 2.8:



```
week9_class_assignment.ipynb x +
colab.research.google.com/drive/1H6uDq6j7ewiXNEDak9Cv0V7NluP_GKks#scrollTo=15lbbASANole
File Edit View Insert Runtime Tools Help
+ Code + Text
[9] print(f"Concatenated along dimension 0: \n{concatenated_dim0}")
print(f"Concatenated along dimension 1: \n{concatenated_dim1}")

Concatenated along dimension 0:
tensor([[[[-1.1353,  1.2150],
          [ 0.7259, -0.5560],
          [ 0.4417,  0.7349],
          [-1.3035, -0.6180]]]])
Concatenated along dimension 1:
tensor([[[[-1.1353,  1.2150,  0.4417,  0.7349],
          [ 0.7259, -0.5560, -1.3035, -0.6180]]]])

# step 2.8
# 1. Create a 3D random tensor x8 of shape (2, 3, 4) with torch.randint.
x8 = torch.randint(0, 10, (2, 3, 4)).float()

# 2. Compute the maximum value in the entire tensor.
maximum_value = x8.max()

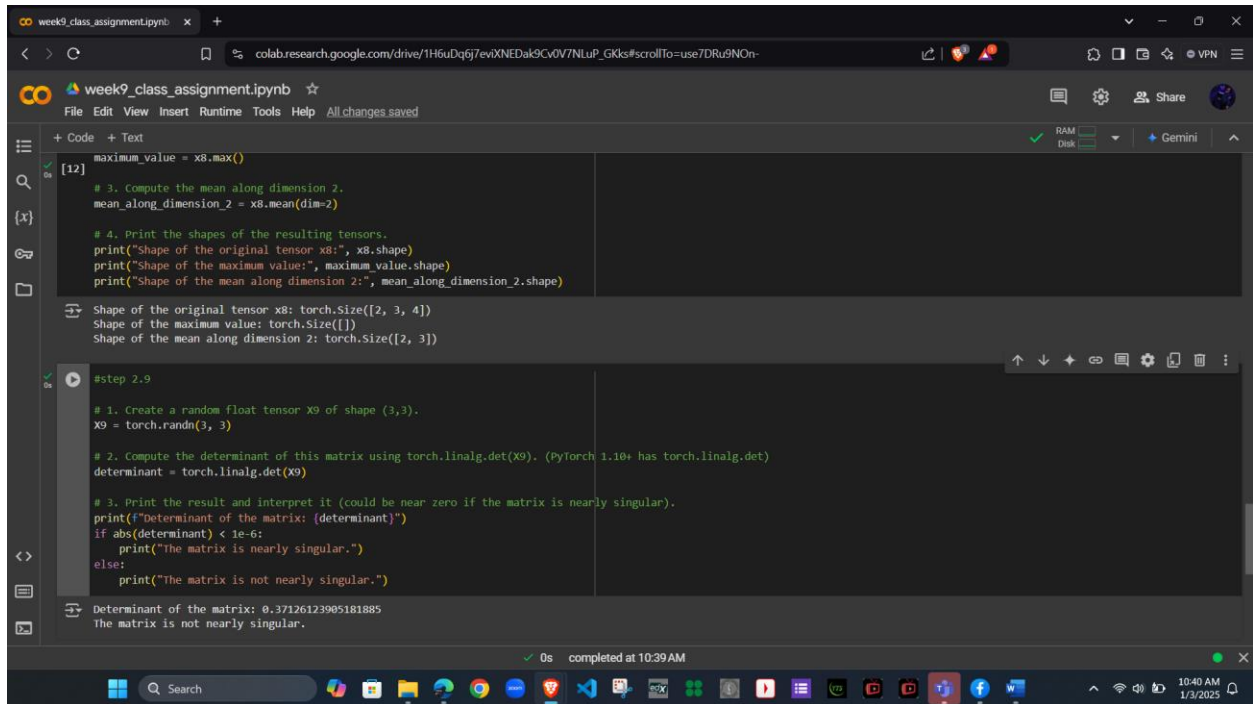
# 3. Compute the mean along dimension 2.
mean_along_dimension_2 = x8.mean(dim=2)

# 4. Print the shapes of the resulting tensors.
print("Shape of the original tensor x8:", x8.shape)
print("Shape of the maximum value:", maximum_value.shape)
print("Shape of the mean along dimension 2:", mean_along_dimension_2.shape)

Shape of the original tensor x8: torch.Size([2, 3, 4])
Shape of the maximum value: torch.Size([])
Shape of the mean along dimension 2: torch.Size([2, 3])
0s completed at 10:38 AM
10:38 AM 1/3/2025
```

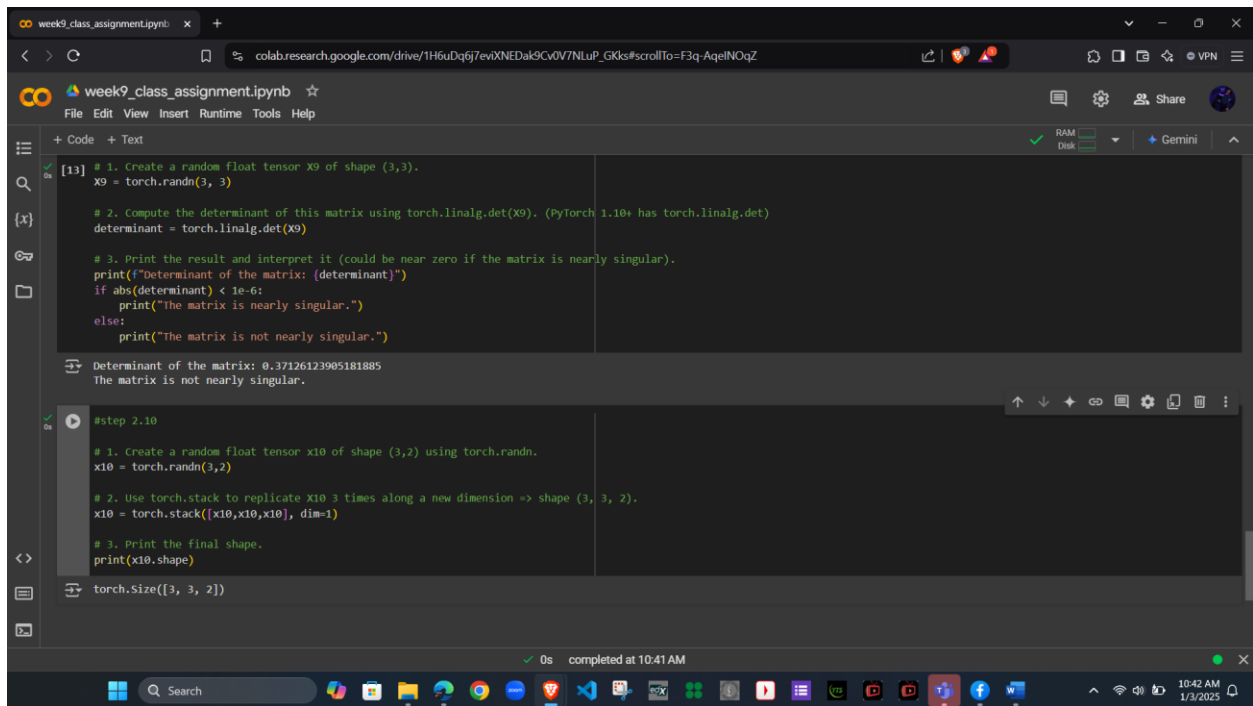


## Step 2.9:



```
week9_class_assignment.ipynb x +
colab.research.google.com/drive/1H6uDq6j7ewXNEDak9Cv0V7NluP_GKks#scrollTo=use7DRu9NOn-
week9_class_assignment.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[12] maximum_value = x8.max()
# 3. Compute the mean along dimension 2.
mean_along_dimension_2 = x8.mean(dim=2)
# 4. Print the shapes of the resulting tensors.
print("Shape of the original tensor x8:", x8.shape)
print("Shape of the maximum value:", maximum_value.shape)
print("Shape of the mean along dimension 2:", mean_along_dimension_2.shape)
Shape of the original tensor x8: torch.Size([2, 3, 4])
Shape of the maximum value: torch.Size([])
Shape of the mean along dimension 2: torch.Size([2, 3])
#step 2.9
# 1. Create a random float tensor x9 of shape (3,3).
x9 = torch.randn(3, 3)
# 2. Compute the determinant of this matrix using torch.linalg.det(x9). (PyTorch 1.10+ has torch.linalg.det)
determinant = torch.linalg.det(x9)
# 3. Print the result and interpret it (could be near zero if the matrix is nearly singular).
print("Determinant of the matrix: (determinant)")
if abs(determinant) < 1e-6:
    print("The matrix is nearly singular.")
else:
    print("The matrix is not nearly singular.")
Determinant of the matrix: 0.37126123905181885
The matrix is not nearly singular.
0s completed at 10:39 AM
10:40 AM 1/3/2025
```

## Step 2.10:



```
# week9_class_assignment.ipynb
colab.research.google.com/drive/1H6uDq6j7ewXNEDak9Cv0V7NLuP_GKks#scrollTo=F3q_AqelNOqZ

File Edit View Insert Runtime Tools Help

+ Code + Text
RAM
Disk
+ Gemini

[11] # 1. Create a random float tensor X9 of shape (3,3).
X9 = torch.randn(3, 3)

# 2. Compute the determinant of this matrix using torch.linalg.det(X9). (PyTorch 1.10+ has torch.linalg.det)
determinant = torch.linalg.det(X9)

# 3. Print the result and interpret it (could be near zero if the matrix is nearly singular).
print(f"Determinant of the matrix: {determinant}")
if abs(determinant) < 1e-6:
    print("The matrix is nearly singular.")
else:
    print("The matrix is not nearly singular.")

Determinant of the matrix: 0.37126123905181885
The matrix is not nearly singular.

#step 2.10

# 1. Create a random float tensor x10 of shape (3,2) using torch.randn.
x10 = torch.randn(3,2)

# 2. Use torch.stack to replicate x10 3 times along a new dimension => shape (3, 3, 2).
x10 = torch.stack([x10,x10,x10], dim=1)

# 3. Print the final shape.
print(x10.shape)

torch.Size([3, 3, 2])

0s completed at 10:41 AM
10:42 AM
1/3/2025
```