```
# !pip install torch torchvision torchaudio
```

Task 1.1: Tạo Tensor

```
import torch
import numpy as np

data = [[1, 2], [3, 4]]

# Tensor từ list
x_data = torch.tensor(data)
print(f"Tensor từ list:\n{x_data}\n")

# Tensor từ NumPy array
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
print(f"Tensor từ NumPy array:\n{x_np}\n")

# Tensor toàn 1, cùng shape với x_data
x_ones = torch.ones_like(x_data)
print(f"Ones Tensor:\n{x_ones}\n")

# Tensor ngẫu nhiên, float
x_rand = torch.rand_like(x_data, dtype=torch.float)
print(f"Random Tensor:\n{x_rand}\n")

# In ra shape, dtype, device
print(f"Shape của tensor: {x_rand.shape}")
print(f"Datatype của tensor: {x_rand.dtype}")
print(f"Device lưu trữ tensor: {x_rand.device}")
```

```
Tensor từ list:
tensor([[1, 2],
        [3, 4]])

Tensor từ NumPy array:
tensor([[1, 2],
        [3, 4]])

Ones Tensor:
tensor([[1, 1],
        [1, 1]])

Random Tensor:
tensor([[0.5773, 0.0550],
        [0.0607, 0.3359]])

Shape của tensor: torch.Size([2, 2])
Datatype của tensor: torch.float32
Device lưu trữ tensor: cpu
```

Task 1.2: Các phép toán trên Tensor

```
# Cộng tensor với chính nó
print(x_data + x_data)

# Nhân tensor với 5
print(x_data * 5)

# Nhân ma trận x_data với x_data.T
print(x_data @ x_data.T)
```

```
tensor([[2, 4],
        [6, 8]])
tensor([[ 5, 10],
        [15, 20]])
tensor([[ 5, 11],
        [11, 25]])
```

Task 1.3: Indexing và Slicing

```
# Hàng đầu tiên
print(x_data[0])

# Cột thứ hai
print(x_data[:, 1])

# Giá trị ở hàng thứ hai, cột thứ hai
print(x_data[1, 1])
```

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

Task 1.4: Thay đổi hình dạng Tensor

```
x = torch.rand(4, 4)
x_reshaped = x.view(16, 1)  #x.reshape(16, 1)
print(x_reshaped.shape)
```

```
torch.Size([16, 1])
```

Task 2.1: Thực hành với autograd

```
x = torch.ones(1, requires_grad=True)
y = x + 2
z = y * y * 3
z.backward()  # dz/dx được tính

print(x.grad)  # 18, vì dz/dx = 6*(x+2), x=1
# Khi tính gradient lần đầu, PyTorch xóa graph tính toán để tiết kiệm bộ nhớ.
# Gọi z.backward() lần nữa sẽ gây lỗi "RuntimeError: Trying to backward through the graph a second time.""
# Nếu muốn tính lại nhiều lần, dùng z.backward(retain_graph=True).
```

```
tensor([18.])
```

Task 3.1: Lớp nn.Linear

```
linear_layer = torch.nn.Linear(in_features=5, out_features=2)
input_tensor = torch.randn(3, 5)
output = linear_layer(input_tensor)

print(input_tensor.shape)  # (3,5)
print(output.shape)        # (3,2)
```

```
torch.Size([3, 5])
torch.Size([3, 2])
```

Task 3.2: Lớp nn.Embedding

```
embedding_layer = torch.nn.Embedding(num_embeddings=10, embedding_dim=3)
input_indices = torch.LongTensor([1, 5, 0, 8])
embeddings = embedding_layer(input_indices)

print(input_indices.shape)  # (4,)
print(embeddings.shape)     # (4,3)
```

```
torch.Size([4])
torch.Size([4, 3])
```

Task 3.3: Kết hợp thành một nn.Module

```
from torch import nn

class MyFirstModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim):
        super(MyFirstModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.linear = nn.Linear(embedding_dim, hidden_dim)
        self.activation = nn.ReLU()
        self.output_layer = nn.Linear(hidden_dim, output_dim)

    def forward(self, indices):
        embeds = self.embedding(indices)
        hidden = self.activation(self.linear(embeds))
        output = self.output_layer(hidden)
        return output

model = MyFirstModel(vocab_size=100, embedding_dim=16, hidden_dim=8, output_dim=2)
input_data = torch.LongTensor([[1, 2, 5, 9]])
output_data = model(input_data)
print(output_data.shape)  # (1,4,2)
```

```
torch.Size([1, 4, 2])
```