WORKING WITH TENSORS

```
import torch
```

2.1 Creating Tensors

```
# From a Python list
data = [1.0, 2.0, 3.0]
tensor_from_list = torch.tensor(data)
print(f"Tensor from list: {tensor_from_list}")

# From a NumPy array
import numpy as np
np_array = np.array(data)
tensor_from_np = torch.tensor(np_array)
print(f"Tensor from NumPy array: {tensor_from_np}")

# Creating a tensor with specific data type
tensor_float32 = torch.tensor(data, dtype=torch.float32)
print(f"Tensor with dtype float32: {tensor_float32}")
```

```
Tensor from list: tensor([1., 2., 3.])
Tensor from NumPy array: tensor([1., 2., 3.], dtype=torch.float64)
Tensor with dtype float32: tensor([1., 2., 3.])
```

2.2 Tensor Initialization Methods

```
# Zero tensor
zero_tensor = torch.zeros(3, 3)
print(f"Zero tensor: \n{zero_tensor}")

# Ones tensor
ones_tensor = torch.ones(3, 3)
print(f"Ones tensor: \n{ones_tensor}")

# Identity matrix
identity_matrix = torch.eye(3)
print(f"Identity matrix: \n{identity_matrix}")

# Random tensor
random_tensor = torch.rand(3, 3)
print(f"Random tensor: \n{random_tensor}")

# Tensor with specific values
specific_tensor = torch.tensor([[1, 2], [3, 4]])
print(f"Tensor with specific values: \n{specific_tensor}")
```

```
Zero tensor:
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]])
Ones tensor:
tensor([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]])
Identity matrix:
tensor([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]])
Random tensor:
tensor([[0.6465, 0.6954, 0.1034],
        [0.5214, 0.8445, 0.0349],
        [0.2748, 0.8276, 0.1770]])
Tensor with specific values:
tensor([[1, 2],
        [3, 4]])
```

2.3 Tensor Operations

```
# Element-wise addition
a = torch.tensor([1.0, 2.0, 3.0])
b = torch.tensor([4.0, 5.0, 6.0])
add_result = a + b
print(f"Element-wise addition: {add_result}")

# Element-wise multiplication
mul_result = a * b
print(f"Element-wise multiplication: {mul_result}")

# Matrix multiplication
matrix_a = torch.tensor([[1, 2], [3, 4]])
matrix_b = torch.tensor([[5, 6], [7, 8]])
matmul_result = torch.matmul(matrix_a, matrix_b)
print(f"Matrix multiplication: \n{matmul_result}")

# Broadcasting
scalar = 2
broadcast_result = a * scalar
print(f"Broadcasting result: {broadcast_result}")
```

```
Element-wise addition: tensor([5., 7., 9.])
Element-wise multiplication: tensor([ 4., 10., 18.])
Matrix multiplication:
tensor([[19, 22],
        [43, 50]])
Broadcasting result: tensor([2., 4., 6.])
```

2.4 Reshaping Tensors

```
# Reshaping a tensor
tensor = torch.tensor([[1, 2, 3], [4, 5, 6]])
reshaped_tensor = tensor.view(3, 2)
print(f"Original tensor: \n{tensor}")
print(f"Reshaped tensor: \n{reshaped_tensor}")

# Flattening a tensor
flattened_tensor = tensor.view(-1)
print(f"Flattened tensor: {flattened_tensor}")

# Transpose a tensor
transposed_tensor = tensor.t()
print(f"Transposed tensor: \n{transposed_tensor}")
```

```
Original tensor:
tensor([[1, 2, 3],
        [4, 5, 6]])
Reshaped tensor:
tensor([[1, 2],
        [3, 4],
        [5, 6]])
Flattened tensor: tensor([1, 2, 3, 4, 5, 6])
Transposed tensor:
tensor([[1, 4],
        [2, 5],
        [3, 6]])
```

2.5 Indexing and Slicing Tensors

```
# Indexing
tensor = torch.tensor([[1, 2, 3], [4, 5, 6]])
print(f"Element at index (0, 1): {tensor[0, 1]}")

# Slicing
sliced_tensor = tensor[:, 1]
print(f"Sliced tensor (all rows, column 1): {sliced_tensor}")

# Advanced indexing
advanced_indexing = tensor[tensor > 3]
print(f"Elements greater than 3: {advanced_indexing}")
```

```
Element at index (0, 1): 2
Sliced tensor (all rows, column 1): tensor([2, 5])
Elements greater than 3: tensor([4, 5, 6])
```

2.6 GPU Support

```
# Check if GPU is available
if torch.cuda.is_available():
    tensor = tensor.to('cuda')
    print(f"Tensor on GPU: {tensor}")
else:
    print("CUDA is not available. Running on CPU.")

# Perform operations on GPU
if torch.cuda.is_available():
    tensor_a = torch.rand(3, 3).to('cuda')
    tensor_b = torch.rand(3, 3).to('cuda')
    result = tensor_a + tensor_b
    print(f"Result of addition on GPU: {result}")

```

```
CUDA is not available. Running on CPU.
```

EXERCISES

1. Tensor Arithmetic: Create two random tensors and perform element-wise addition, subtraction, multiplication, and division. Calculate the dot product of two tensors.
2. Tensor Reshaping: Create a 3x3 tensor and reshape it into a 1D tensor (flatten). Create a 2D tensor and transpose it.
3. Indexing and Slicing: Create a 4x4 tensor and extract a 2x2 sub-tensor from it. Use advanced indexing to extract all elements greater than a certain value from a tensor.
4. GPU Computation: Move a tensor to GPU and perform a basic arithmetic operation on it. Measure the time difference between performing an operation on CPU and GPU.