Lab Sheet: Introduction to Python and PyTorch

Lab Duration: 2-3 hours

Lab Section 1: Let's Dive into Python Magic! 🐍✨

Objective:Welcome to the world of Python! In this section, we will cover some fundamental concepts of Python programming. Whether you are a beginner or looking for a quick refresher, these exercises will help you get comfortable with the basics.

Task 1: Setting up our Workspace

1.1 **Install Python:**

Python is the magical language we'll be casting spells with. Visit **python.org** to install it on your computer. Don't worry; it's as easy as waving a wand!

1.2 **Install Jupyter Notebook:**

Jupyter Notebook is our magical parchment for writing Python spells. Open your command prompt and cast the spell: `pip install notebook`. Now, your notebook is ready to capture your magical code!

Task 2: Unveiling the Secrets of Python Spells

2.1 **Let's Open the Spellbook (Jupyter Notebook):**

In your command prompt, type jupyter notebook to open your spellbook. Click on **New** and choose "**Python 3**" to create your first spell page.

2.2 **Learn the Language of Magic:**

Welcome to the magical realm of Python! We'll start with the basics:

- Variables and Data Types
- Arithmetic Operations
- String Manipulation

- Lists and Indexing
- Control Flow

Magical Exercises:

Exercise 1: Variables and Data Types

Objective: Understand variables and data types in Python.

1. Create a variable named my_variable and assign it the value 42.

   Hint: Python Solution

```
my_variable = 42
```

2. Print the value of my_variable to the console.

   Hint: Python Solution

```
print(my_variable)
```

3. Create variables of different data types: integer, float, string, and boolean.

   Hint: Python Solution

```
int_variable = 10
float_variable = 3.14
string_variable = "Hello, Python!"
bool_variable = True
```

4. Print the data types of the variables.

   Hint: Python Solution

```
print(type(int_variable))
print(type(float_variable))
print(type(string_variable))
print(type(bool_variable))
```

Exercise 2: Control Flow

Objective: Learn about control flow structures in Python.

1. Create an if-else statement to check if a number is even or odd.

Hint: Python Solution

```python
number = 15
if number % 2 == 0:
    print("Even")
else:
    print("Odd")
```

2. Use a for loop to iterate through a list of fruits and print each one.

Hint: Python Solution

```python
counter = 1
while counter <= 5:
    print(counter)
    counter += 1
```

3. Create a while loop that prints numbers from 1 to 5.
Hint: Python Solution

```python
counter = 1
while counter <= 5:
    print(counter)
    counter += 1
```

Exercise 3: Functions

Objective: Define and call functions in Python.

1. Create a function named greet_user that prints a welcome message.

Hint: Python Solution

```python
def greet_user():
    print("Welcome to Python!")

greet_user()
```

2. Define a function add_numbers that takes two parameters and returns their sum.

Hint: Python Solution

```python
def add_numbers(a, b):
```

```
    return a + b

result = add_numbers(5, 7)
print(result)
```

3. Create a function is_even that returns True if a number is even and False otherwise.

   Hint: Python Solution

```
def is_even(number):
    return number % 2 == 0

print(is_even(10))
print(is_even(7))
```

Exercise 4: lists

1. Create a list named my_list containing integers.

   Hint: Python Solution

```
my_list = [1, 2, 3, 4, 5]
```

2. Access and print the third element of the list.

   Hint: Python Solution

```
print(my_list[2])
```

3. Modify the second element of the list and print the updated list.

   Hint: Python Solution

```
my_list[1] = 10
print(my_list)
```

4. Add a new element to the end of the list and print the modified list.

   Hint: Python Solution

```
my_list.append(6)
print(my_list)
```

5. Use list slicing to create a new list containing only the first three elements.

   Hint: Python Solution

```
new_list = my_list[:3]
print(new_list)
```

6. Check if a specific element exists in the list.

   Hint: Python Solution

```
element_to_check = 8
if element_to_check in my_list:
    print(f"{element_to_check} exists in the list.")
else:
    print(f"{element_to_check} does not exist in the list.")
```

Exercise 1: The Enchanted List

Write a Python spell to find the sum of all even numbers in a list. Imagine the list is a magical collection of gems; your spell should reveal the sum of the enchanted even gems.

```
# Creating a magical list of gems
enchanted_list = [2, 5, 8, 10, 3, 6, 12, 7]

# Casting the spell to find the sum of even gems
sum_of_even_gems = sum(num for num in enchanted_list if num % 2 == 0)

# Revealing the magical result
print("The sum of even gems is:", sum_of_even_gems)
```

Hint: Python lists

Exercise 2: The Palindrome Quest

Embark on a quest to create a spell that checks if a given word, is a palindrome. Is it a magical word that reads the same backward as forward?
Hint: A palindrome, like "level" or "radar," is a magical word indeed!

Hint: Python lists

```
# Creating a magical word
magical_word = "level"

# Casting the spell to check if it's a palindrome
is_palindrome = magical_word == magical_word[::-1]

# Revealing the magical result
if is_palindrome:
    print(f"{magical_word} is a magical palindrome!")
else:
    print(f"{magical_word} is not a magical palindrome.")
```

Lab Section 2: Introduction to NumPy and Pandas

Objective: Gain familiarity with NumPy and Pandas libraries for numerical and data manipulation in Python. You can visit this video to learn about NumPy. Here are just a few short exercises to explain the basics.

Exercise 1: NumPy Basics

1. Import the NumPy library.

   Hint: Python Solution

```
import numpy as np
```

2. Create a NumPy array named my_array with integers from 1 to 5.

   Hint: Python Solution

```
my_array = np.array([1, 2, 3, 4, 5])
```

3. Perform basic operations on the array, such as finding the sum, mean, and maximum value.

Hint: Python Solution

```
array_sum = np.sum(my_array)
array_mean = np.mean(my_array)
array_max = np.max(my_array)
print(f"Sum: {array_sum}, Mean: {array_mean}, Max: {array_max}")
```

4. Create a 2D NumPy array and perform operations like finding the transpose.

Hint: Python Solution

```
my_2d_array = np.array([[1, 2, 3], [4, 5, 6]])
transposed_array = np.transpose(my_2d_array)
print(f"Original Array:\n{my_2d_array}\nTransposed
Array:\n{transposed_array}")
```

Exercise 2: Introduction to Pandas

1. Import the Pandas library.

```
Hint: Python Solution
import pandas as pd
```

2. Create a Pandas Series and DataFrame from Python lists.

Hint: Python Solution

```
series_data = pd.Series([10, 20, 30, 40, 50])
dataframe_data = pd.DataFrame({'Name': ['Alice', 'Bob', 'Charlie'], 'Age':
[25, 30, 35]})
print(f"Series:\n{series_data}\n\nDataFrame:\n{dataframe_data}")
```

3. Load a sample dataset using Pandas (e.g., Iris dataset).

Hint: Python Solution

```
iris_dataset = pd.read_csv('https://archive.ics.uci.edu/ml/machine-
learning-databases/iris/iris.data', header=None)
print(f"Iris Dataset:\n{iris_dataset.head()}")
```

4. Perform basic operations on the DataFrame, such as selecting specific columns and filtering data.

Hint: Python Solution

```
selected_columns = dataframe_data[['Name']]
filtered_data = dataframe_data[dataframe_data['Age'] > 30]
print(f"Selected Columns:\n{selected_columns}\nFiltered
Data:\n{filtered_data}")
```

These exercises will introduce you to the fundamental concepts of NumPy and Pandas, essential libraries for data manipulation and analysis in Python. Explore additional functionalities and challenges to deepen your understanding. Happy coding!

Lab Section 3: Introduction to PyTorch Basics

Task 1: Installing PyTorch

Open a new Python environment or Jupyter notebook. Install PyTorch by running the following command:

```
pip install torch torchvision
```

Task 2: PyTorch Tensors

1. Import the PyTorch library:

```
import torch
```

2. Create a PyTorch tensor with values 1, 2, 3:

```
tensor = torch.tensor([1, 2, 3])
print(tensor)
```

3. Check the type and size of the tensor:

```
print(type(tensor))
print(tensor.size())
```

Task 3: Tensor Operations

1. Perform basic tensor operations:

```
# Addition
result_add = tensor + 5
print("Addition:", result_add)

# Multiplication
result_mul = tensor * 2
print("Multiplication:", result_mul)
```

2. Create a second tensor and perform element-wise multiplication:

```
tensor2 = torch.tensor([2, 4, 6])
result_mul = tensor * tensor2
print("Element-wise Multiplication:", result_mul)
```

Task 4: PyTorch Modules and Operations

In PyTorch, modules are a way to organize and encapsulate reusable pieces of code. They provide a convenient method to define, store, and operate on parameters during training.

1. Create a Simple Module:

```
import torch
import torch.nn as nn

class SimpleModule(nn.Module):
    def __init__(self):
        super(SimpleModule, self).__init__()
        self.weight = nn.Parameter(torch.randn(5, 5))
        self.bias = nn.Parameter(torch.zeros(5))

    def forward(self, x):
        return torch.matmul(x, self.weight) + self.bias
```

Here, we define a simple module with a trainable weight matrix and bias vector.

2. Instantiate the Module:

```
my_module = SimpleModule()
```

3. Create an instance of the module and perform operations:

```
input_data = torch.randn(3, 5)
output_data = my_module(input_data)
```

4. Pass input data through the module to get output data.

   Viewing the Results:

```
print("Input Data:", input_data)
print("Output Data:", output_data)
```

   Print the input and output data to observe the transformation.

Understanding PyTorch modules is fundamental for building and working with more complex neural network architectures.

Task 5: Loading and Displaying Images

install the matplotlib library:

```
pip install matplotlib
```

Load and display a sample image using PyTorch:

```
from torchvision import datasets, transforms
from matplotlib import pyplot as plt

# Download and load a sample image
data_transform = transforms.Compose([transforms.ToTensor()])
dataset = datasets.FakeData(transform=data_transform)
dataloader = torch.utils.data.DataLoader(dataset, batch_size=1,
shuffle=True)

# Display the image
for img, _ in dataloader:
    plt.imshow(img[0].permute(1, 2, 0))
    plt.show()
```

This exercise introduces students to the basics of PyTorch. Familiarizing yourself with these fundamentals is crucial before diving into more advanced topics like neural networks.

Exercise!!

1. Solve all of the exercises above & organize the solutions in a jupyter notebook, which you will need to submit.
2. Additionally (**in the same notebook)**, attempt this simple problem based on pytorch basics.

Problem Statement:

In this exercise, you'll explore fundamental concepts in PyTorch, including basic tensor operations.

Tensor Operations:

- Create two PyTorch tensors (a and b) of shape (2, 3) containing random numerical values.
- Perform basic tensor operations:
- Addition (c = a + b)
- Element-wise multiplication (d = a * b)
- Matrix multiplication (e = a.mm(b.t()))
- Print the results.

Indexing and Slicing:

- Create a PyTorch tensor x with values from 0 to 9.
- Use indexing and slicing to obtain:
- The first element of x.
- The last element of x.
- Elements from index 3 to 7 (inclusive).
- Print the results.

Reshaping:

- Create a PyTorch tensor y of shape (3, 4).
- Reshape y into a tensor of shape (4, 3).
- Print the reshaped tensor.

Guidelines:

- Use PyTorch tensors for all operations.
- Ensure that you understand the PyTorch documentation related to tensor operations.
- Provide comments in your code to explain each step.