The primary goal of this tutorial is to familiarize students with Python while working with Linked List structures. Linked Lists are an essential data structure used in computing to efficiently manage dynamic collections of data. In this tutorial, you will:

- Learn how to install Python and set up a programming environment.
- Understand how to write and execute Python code.
- Implement basic operations on Linked Lists using Python.
- Analyze the performance of Linked List operations.

By the end of this tutorial, you will be comfortable using Python for data structure implementations and performing basic manipulations on Linked Lists.

Before working with Linked Lists in Python, we need to set up our development environment.

Step 1: Check if Python is Installed

To check if Python is already installed, open a terminal or command prompt and type:

```
python --version
```

or

```
python3 --version
```

If Python is installed, you should see an output similar to:

```
Python 3.x.x
```

If Python is not installed, proceed to the next step.

Step 2: Download and Install Python

1. Visit the official Python website: `https://www.python.org/downloads/`
2. Download the latest **Python 3.x.x** version for your operating system.
3. Run the installer:
   - **Windows:** Check the box that says **"Add Python to PATH"** before clicking **Install Now**.
   - **macOS/Linux:** Follow the on-screen instructions.

After installation, verify Python again by running:

```
python --version
```

Now that Python is installed, let's write and execute our first program.

Step 3: Writing a Simple Python Script

Create a file named `hello.py` and add the following code:

```python
print("Hello world!")
```

Step 4: Running the Script

Using Terminal or Command Prompt:

Navigate to the directory where `hello.py` is saved and run:

```
python hello.py
```

or (on some systems):

```
python3 hello.py
```

Expected Output:

```
Hello world!
```

Warm-up

Problem 1. Implement a Python class called `Node` that represents a single node in a **singly linked list**. Each node should contain:

- An integer value `data`
- A reference to the next node `next`

Problem 2. Write a function `traverse(head)` that prints all elements of a **singly linked list**, starting from the **head**.

Problem 3. Implement a Python class called `DoublyNode` that represents a single node in a **doubly linked list**. Each node should contain:

- An integer value `data`
- A reference to the next node `next`
- A reference to the previous node `prev`

Problem 4. Write a function `insert_at_beginning(head, new_data)` that inserts a new node at the **beginning** of a singly linked list.

Problem solving

Problem 5. Write a function `delete_node(head, key)` that removes a node from a **doubly linked list**, given its value.

Problem 6. Given a singly linked list, we would like to traverse the elements of the list in reverse order.

a) Design an algorithm that uses $O(1)$ extra space. What is the best time complexity you can get?

b) Design an algorithm that uses $O(\sqrt{n})$ extra space. What is the best time complexity you can get?

You are not allowed to modify the list, but you are allowed to use position/cursors to move around the list.

Problem 7. Consider the problem of given an integer n, generating all possible permutations of the numbers $\{1, 2, \ldots, n\}$. Provide a recursive algorithm for this problem.