**Final Project Instructions for Basic Python Course**

**Project Name: wit - Version Control System using Click**

In this project, you will develop a simple version control system called **wit**, similar to Git. The system will allow users to manage file versions, add files for tracking, commit changes, and view the version history. All commands will be executed through a Command-Line Interface (CLI) using the **Click** library.

This project **must** be implemented using **Object-Oriented Programming (OOP)** principles. The code should be structured using classes and objects to model the system and its functionality.

**Project Goal:**

The task is to develop a version control system with basic functionality like Git, using Python and Click to write the CLI. All commands will be executed through the command line, and users will be able to perform tasks like creating a repository, adding files, committing changes, and viewing version history.

You should organize the system using OOP to model key entities such as repositories, commits, and files.

**Command Requirements:**

1. **wit init**  
   The wit init command will create a new folder called .wit in the current directory, if such a folder does not already exist. If the .wit folder already exists, an appropriate error message will be displayed.
2. **wit add**   
   The wit add <file> command will add selected files for tracking in the version. This means the files will be included in the next commit.
3. **wit commit -m "message"**  
   The wit commit -m "message" command will create a new version in the repository and save the tracked files with the commit message explaining the changes made.
4. **wit log**  
   The wit log command will display the version history of the system, including the commit hash, date, and commit message for each version that has been saved.
5. **wit status**  
   The wit status command will check if there are any uncommitted changes in the repository (e.g., files that have been modified but not yet committed). If such changes exist, an appropriate message will be displayed.
6. **wit checkout <commit\_id>**  
   The wit checkout <commit\_id> command will allow the user to revert to a previous version by specifying a commit ID. After executing this command, the contents of the files will be updated to the selected version.

**Using Click:**

This project must be implemented using the **Click** library for creating the Command-Line Interface (CLI). The Click library provides easy and efficient tools for defining command-line commands with options and parameters, and it simplifies user interaction. You will use Click to define commands such as wit init, wit add, wit commit, etc., and enable users to interact with the system intuitively via the command line.

**Object-Oriented Programming (OOP):**

The project must be implemented using **Object-Oriented Programming (OOP)** principles. This means that you should use classes and objects to model the version control system. Be sure to organize the code in a way that adheres to OOP concepts such as encapsulation, inheritance, and abstraction.

**Technical Requirements:**

* You need to create a .wit folder to store all version-related information, including commits and other data.
* You must manage files and versions in such a way that allows users to revert to previous versions at any time.
* Each command should accept user input via the CLI and respond appropriately (e.g., create folders, add files for tracking, save versions with commit messages, etc.).
* Use data structures such as dictionaries or lists to manage the version and commit information.

**Expected Outcome at the End of the Project:**

By the end of the project, you will have a complete CLI tool that allows simple version control. Users will be able to add files, create versions, view the history, revert to previous versions, and manage all the basic functions associated with version control.

Since this project is implemented using OOP principles, the code should be well-structured, reusable, and modular.

### Bonus - NTH: Directory Management

For an advanced version of the version control system, you can implement **directory management**. This feature will allow the system to handle file structures, enabling users to organize files within directories, create new directories, and track changes within these directories. The system should be able to represent a hierarchy of directories and files, ensuring that changes such as file creation, deletion, or movement between directories are tracked properly. By managing directories and their contents, the version control system can provide a more flexible and scalable way to organize and track files, making it easier for users to work with large projects that involve multiple directories and subdirectories.

Good luck!