1. Only Pull and Push
    1. Fork my `Github_Collaboration_Demo` repo (for students' practice)
    2. Use the terminal to create two separate cloned repositories:
        1. `git clone https://github.com/cty20010831/Github_Collaboration_Demo.git
    3. First, add `print("Simple Addition Program")` in one cloned repo and push it to origin main
    4. Then, without pulling, add `print("Please see the results")` under the function doc string on the second local repo
    5. If you try to push the changes (and commits) in the second local repo to origin main, it will throw an error message regarding diverging branches
        1. One good practice is always to git fetch (to check for updates on origin main)!
        2. The error message would look like: "**Can't push refs to remote. Try running "Pull" first to integrate your changes.**"
        3. How to solve this problem?
            1. Option 1: You do the merge or rebase "style" of `git pull` instead of the default fast-forward "style". Suppose you do the normal merge "style" (as what we did in class, but feel free to try the rebase "style") `git pull --no-rebase origin main`, a conflict solver file will be opened on your code editor to resolve the conflict.
                1. Side note: `git fetch origin main` and `git merge origin main` will achieve the same result
            2. Option 2: You "withdraw" the action using `git reset --soft HEAD~1` then you either do merge/rebase after you commit your change (but do not push it to origin main) => E.g., `git merge origin main`, this will open a conflict solver file on your code editor to resolve the conflict.
        4. Note: for resolving merge conflicts, you actually have four choices: 1) accepting incoming changes, 2) accepting ongoing changes, 3) accept both changes, and 4) accept none.
2. Fork and Pull Request (see the screenshots of steps I included in lab powerpoint for more details)
    1. Create a fork of the upstream repo (in this case, https://github.com/cty20010831/Github_Collaboration_Demo.git)
    2. (Optional) Set local repository of the forked repository
        1. If you choose to do so, do remember to push and pull your changes between you local forked repo and forked repo on origin
    3. Commit changes in the forked repository (or commit and push changes if in local forked repository)

4. Open pull request
    1. You can either click "Contribute" button on Github or click the "Pull Request" button, both of which opens the pull request window
5. Create pull request
6. Wait for the response from the owner of the upstream repository (either accept or decline)
    1. Please don't actually send me the pull request !!!

3. Branch and Merge
    1. Create two feature branches ("feature_1" and "feature_2")
        1. These branches can be created either directly on origin (i.e., on Github) or locally (I will demonstrate one created on origin and the other locally in the lab)
            1. If created on origin, you can:
                1. Either fetch all new branches from GitHub: `git fetch origin`
                2. Or create & track the remote branch locally (but do remember to fetch all remote branches!): `git checkout -b feature_1 origin/feature_1
            2. If created locally:
                1. `git checkout –b feature_2; git push origin feature_2`
                2. Note: you can use `git push ––set–upstream origin feature_2` to avoid manually push feature_2 after local creation
        2. These branches will be handled by two fake users as in "Only Pull and Push"
        3. (After this step) Ensure that `1_Clone_and_Push` and `2_Fork_and_Pull_Request` stay on "main" branch when doing demonstration of the previous two approaches of collaboration
    2. Begin with an example with no merge conflicts => Both branches create a new (yet different) file and merge to main branch later (use `git switch` or `git checkout` to switch branches: "feature_1" branch: add a txt file called "feature_1" & "feature_2" branch : add a txt file called "feature_2"
        1. Demo 1 (Recommend approach): Use feature branch workflow to have pull requests after pushing changes in local branches to their respective branches (i.e., Solve **on GitHub** by creating PRs and using GitHub's UI to resolve conflicts)
        2. Demo 2: Merge changes onto local main branch, meaning that solving the conflicts locally, usually not recommended
            1. Commit the changes on local branch (do not push it to origin!)
            2. Switch to main branch and try to merge the branch to main branch
            3. Then push to origin: `git push origin main`
    3. Next, showcase an example with merge conflicts => Both branches edit the same part of the same file (specifically, add different comments under the comment "//

Function to add two numbers" in line 4 of test.cpp)

1. Demo 1 (Recommend approach): Use feature branch workflow to have pull requests after pushing changes in local branches to their respective branches (i.e., Solve **on GitHub** by creating PRs and using GitHub's UI to resolve conflicts)
    1. The only difference here is that compared to the case when there is no merge conflicts, you will receive a merge conflict warning message when you deal with the second pull request (the order does not matter). You can now solve the merge conflict on Github (sharing the same principles of solving merge conflicts as you do locally).
2. Demo 2: Merge changes onto local main branch, meaning that solving the conflicts locally, usually not recommended
    1. Commit the changes on local branch (do not push it to origin!)
    2. Switch to main branch and try to merge the branch to main branch
    3. Actually you can track the difference between branches first using `git diff <branch_name> <branch_name> <branch_name>` (however number of branches you want)
    4. In this case, there will be merge conflicts that need to be resolved when both branches are merged to the main branch
    5. After resolving the conflict locally, `git push origin main`
    6. Alternative conflict resolution methods: Use a **rebase workflow** instead of merging: `git checkout feature_1` & `git rebase main` (this forces feature_1 to be updated before merging)
4. (Optional) You can also try deleting branches (if you want to play back again)
    1. Option 1: First delete the local branch `git branch -d <branch_name>` and then delete Remote Branch (`git push origin --delete <branch_name>`)
    2. Option 2: First delete the branch on Github, and then update the deletion of remote branch locally (`git fetch --prune`)
5. (Optional) A taste of three-way merge (aside from fast-forward merge)
    1. We will again add different comments under the comment "// Function to add two numbers" in line 4 of test.cpp (feel free to add different stuff) for main and feature_1
    2. When you commit the change in main branch and then commit the change in feature_1 branch, and then try to merge feature_1 to main (`git merge feature_1` on main branch), you will get a three-way merge.
    3. Even if there are no merge conflicts, Git will still perform a three-way merge and create a merge commit. Feel free to test that out (when there are no conflicts and when there are)!