Github and Code Cleaning:

- Github vs Git:
    - GitHub --> [browser]
        - GitHub is a website that allows you to upload your git repositories online. It allows you to have a backup of your files online, has a visual interface to navigate your repos, and it allows other people to be able to view, copy, or contribute to your repos.
    - Git --> [terminal]
        - Git is a version control system. It allows you to manage the history of your git repositories.
- Motivation:
    - For each codebase (repository) I own, I want to write code where:
        - 1.Progress loss is minimized
        - 2.Iterating on different versions of the code is easy
        - 3.Collaboration is productive
- Minimal Progress Loss:
    - This is achieved by effectively “backing up your work”.
    - By creating regular save points (called commits) and pushing them up to GitHub (from your laptop), if your laptop is destroyed you will only lose the commits you did not upload to GitHub.
- Iterating on Different Versions:
    - The ease or difficulty of adding a new feature to the code base may depend on the state / version of the codebase.
    - It may be easiest to add this feature at a specific commit.
- Looks like we need:
    - 1.A way to preserve both versions of history
    - 2.A way to overwrite history if we choose (this is dangerous as we will lose that history)
- Branch off a particular commit:
- Push commits per branch
- Creates lots of branches
- But one branch needs to chosen as the primary, stable branch
- This branch is typically called the “master” or “main” branch
- Other branches are usually named after either the feature that is being developed on or the major or minor version of the software / product
- At some point we will want to clean up certain branches by merging them with the master / main branch or with each other.
- Merging is trivial if the base of one branch is the head of the other - the changes are “simply” appended.
- When this is not the case, commits can conflict with each other

- We need to change the base of the login-page branch (rebase) to be at the head of the master branch
- This is not a simple operation! It will often require manual intervention to resolve the conflicts.

Collaboration: Possible to collaborate on a single repository by having each collaborator create new branches for development and having a process for merging their branch into master.This request to merge code is done by a Pull Request (PR).

- Typically (in particular with open source repositories), repository owners prefer not having to manage collaborator permissions on the repository.
    - In order to contribute code, collaborators must:
        - 1.Make a copy of (fork) the main repository
        - 2.Make all the changes they want to this copy
        - 3.Request that part of their copy be merged into the main repository via a Pull Request (PR)
- In the time it took you to implement your new feature, the main repository has changed since the time you forked the repository.
- How do you keep your copy up to date?
- You could delete your copy and re-copy the latest… But you would lose all the work you committed to your copy!
- Luckily, we’re only interested in keeping the master branch in sync! Why?
- Fetch + merge = pull
    - This is trivial when the base of one branch matches the head of the other.
    - If you never commit anything to your master branch, keeping your master branch in sync with the main repositories is easy! And keeping all branches attached comes for free!
    - As a rule, always create a new branch when developing - never commit directly to the master branch

Clean Code:
- “Software Systems get replaced not when they wear out but when they crumble under their own weight because they have become too complex”

Gems of Clean Code:
- Structure
    - Before writing code ask “How will someone use this (or part of this) code?”. Minimize side effects
    - Do one thing
- ●Method
    - Top Down Approach
    - Bottom up Approach

    - Solve the problem first - then improve / refine
- General
    - Boring code is good code: keep it simple
    - Late Binding: start vague and refine (don’t commit to specificity)
    - Many functions with small bodies > one function with large body
    - Check soundness by reading your code before testing
- Functional Programming:
    - “Functional Programs are mathematical expressions that are evaluated and reasoned about much like ordinary mathematical functions. As a result, these expressions are simple to analyze and compose for large-scale programs”
- Example: max path sum
    - Starting at the top of the triangle and moving down to adjacent numbers below: Find the path from the root to a leaf with the maximum sum.
        - **max**(**foldleft**(triangle, [], **lam**(xs, acc) **=>** **myfold**(xs, acc)))
        - **myfold**(xs, acc) :
        -             option1 = **map2**([0, acc...], xs, **lam**(x, y) **=>** x + y)
        -             option2 = **map2**([acc…, 0], xs, **lam**(x, y) **=>** x + y)
        -   **return map2**(option1, option2, **lam**(x, y) **=> if** x > y **then** x **else** y)
- Python example:
    - funcs = []
    - results = []
    - **for** x **in** range(**3**):
    -     **def some_func**():
    -             **return** x
    -         funcs.append(some_func)
    -     results.append(some_func()) # note the function call here
    - funcs_results = [func() **for** func **in** funcs]
    - **print**(results) # [0,1,2]
    - **print**(funcs_results)
    -