Week 9 - Reproducible code and addressing code reviews

Based on the code reviews that you received from your student assistants, today is the time to improve your GitHub repositories and code structure! The following list summarizes the most common errors that were made in the first weeks and explains how to correct them. Please go over them, and you'll soon find yourself on the way to a clean and functional GitHub coding project with Python! Try to incorporate everything you learn today directly into your BIO-210 project on GitHub (for all releases starting with v3; if you already released it then release an update, e.g. v3.1)!

Make sure to release your code by 10am next Monday (release notes)!

1. **Addressing code reviews (issues)**

   The student assistants opened various issues during their review, make sure to go through them carefully and address each one in your next commits. This is an important part of maintaining a clear development history and showing how you respond to feedback. Some good approaches are:

   - When committing your changes, link them to the corresponding issue to indicate what problem you are addressing. You can do this directly in your commit message by writing `git commit -m "closes #3"`, which will literally close issue #3 once you push to GitHub. Example: `https://github.com/DeepLabCut/DeepLabCut/issues/1150`
   - You can also cross-link issues and pull requests by including hyperlinks or references. Example: `https://github.com/DeepLabCut/DeepLabCut/pull/1582`
   - To mention an issue without closing it (for example, when you are only partially addressing it), use a keyword such as `related to`, `ref`, or simply the issue number (e.g., `#3`) instead of closing keywords like `closes`, `fixes`, or `resolves`. This creates a link between the commit or pull request and the issue, but does not automatically close the issue.

2. **How to exclude file types from git repository**

   Add all file types to the `.gitignore` file that you don't want to track (e.g., notebook checkpoints, pycache, files from your IDE, virtual environments). See the documentation at `https://git-scm.com/docs/gitignore` and also check out this example: `https://github.com/EPFL-BIO-210/demo-project/blob/main/.gitignore`.

   Note that if you add the `.gitignore` file *after* files that you want to ignore are tracked, then you need to "manually" remove them to also get rid of them (i.e. delete them, then stage and commit; you can also manually untrack with `git rm --cached <file>`).

   You can also find a wide rande of ready-made `.gitignore` templates for different environments and tools here: `https://github.com/github/gitignore`.

3. **How to organize your Python environments (with conda)**

   For reproducibility, keeping track of dependencies is important. A simple way to achieve this is by using conda environments. You can create one isolated Python environment for each project and install all required dependencies within it.

   Go through the following guide `https://conda.io/projects/conda/en/latest/user-guide/getting-started.html` for conda environments, and review the Anaconda tutorial from Week 3: `https://github.com/EPFL-BIO-210/BIO-210-CourseMaterials/blob/main/docs/week_3.md`. To make the environment you have created for your project reproducible, export it to an `environment.yml` file. A guide for that can be found here: `https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#sharing-an-environment`.

4. **What to write in a good README file?**

   A README is a text file that introduces and explains your GitHub project. It should provide enough information for others to understand what the project is about and how to run your code. Read this post https://www.freecodecamp.org/news/how-to-write-a-good-readme-file/ for what to include in a good README.

   README files are written in Markdown, which allows you to write nicely formatted and visually appealing texts. See this guide for basic syntax and formatting: https://docs.github.com/en/github/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax.

   You can also get inspiration from the following example projects: https://github.com/DeepLabCut/DeepLabCut/blob/master/README.md, https://github.com/EPFL-BIO-210/demo-project and https://github.com/psf/black.

Search for different public GitHub repositories on your own, compare them, and find out what makes a good and clean structure that helps for reproducible code and experiments!