Multivariable Calculus Self-Learning Module

Setup & Usage Manual

Contents

List of Figures

1   Introduction

The website is currently hosted for free by *GitHub Pages*. In order to deploy a website using GitHub Pages you need a *GitHub* account.

What is GitHub?   GitHub is a popular, free, cloud-based platform that allows users to store and share files. It is mainly (but not exclusively) used for collaboration on coding projects. Think of it as a storage space just like Google Drive for anything you might want to share with others. Each project on GitHub is stored in its own directory, which is called a *repository*. GitHub uses *Git* to keep track of any changes that happen in a repository.

What is Git?   Git is an open-source version control software which is widely used to keep track of any changes that happen in a file. Think of it as an alternative to making multiple files such as: "exam-v1", "exam-v2", "exam-v2-fixed", ..., "exam-final". Git keeps track of all the different versions for you, and it lets you revert back to any previous version of your file, or test stuff by creating a new version branch without ever affecting your main branch. GitHub uses Git, but Git is independent of GitHub and it works with its own local repositories on your computer.

How does GitHub Pages work?   You simply deploy your website directly from your GitHub repository on GitHub Pages. After that point, your website is always running, and whenever you make a change to the repository, it will be implemented almost immediately on the live website.

What is the point of all this?   The idea is the following:

1. You keep your own version of the website on your computer, along with a local installation of Git.

2. Say you make some changes that you are happy with, e.g. you add an exercise or fix a mistake. You *commit* the changes to your local Git repository.

3. You then *push* the local Git repository to the online GitHub repository.

4. GitHub Pages will automatically detect the new changes from your GitHub repository and deploy a new version of your website.

The good part is that once set up this process is totally free, standardised, relatively easy, and should be familiar to anyone who has ever collaborated on a coding project.

2 Setup

Setup is a bit long but at least you only have to do it once.

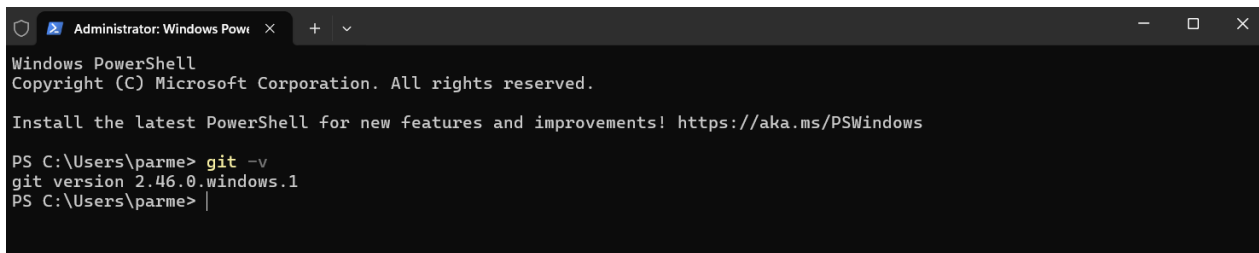### 2.1 Git

The latest version of Git for Windows can be downloaded from:

`https://git-scm.com/download/win`

In the vast majority of cases the 64-bit version (first link) is required. The default installation options are fine. After installation is done, check if Git has been installed correctly by opening a terminal (can be done by *Windows key + X > Terminal (Admin)*) and typing the command:

`git -v`

The response should be the latest version of Git (e.g. Figure 1).

Figure 1: `git -v`

If not, try restarting the computer and check again. Restart is sometimes needed so that folder in which Git is installed is permanently added to the Windows PATH variable, which lets Windows know what the `git` command means. If this also doesn't work, Git must be added to the PATH manually. This is annoying and should (and most likely will) not happen, and at that point it is probably best to Google something like "add git to path windows 11" (or whatever your Windows version is) and follow the detailed instructions.
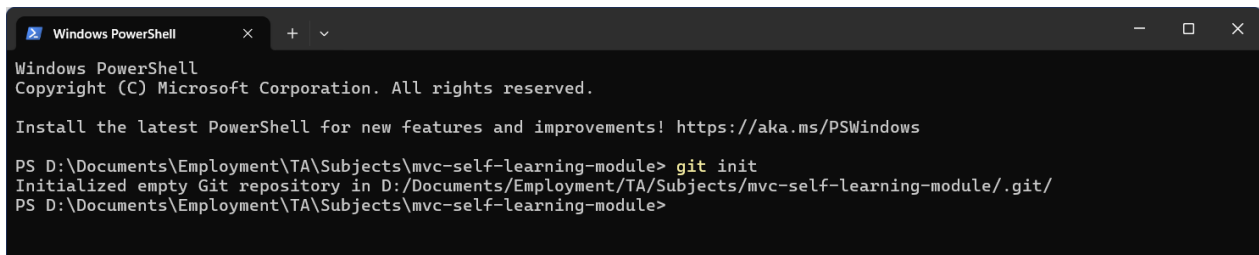
2.1.1 Initiating a local Git repository

Assuming that Git is already installed, there are two ways to initiate a local Git repository.

Option 1: Initiate a new Git repository for an existing local directory.   Suppose you have the directory *mvc-self-learning-module* somewhere on your computer and it is not already associated with a local Git repository. In order to initiate one, you can open the directory in a terminal (can be done by navigating within the directory using the File Explorer, and then on any empty space *Right click > Open in Terminal*) and then type the command:

`git init`

The output should be something like Figure 2.

Figure 2: `git init`

You can visually inspect the Git repository if you enable the option of seeing the hidden files on your computer (can be switched on within the File Explorer itself from *View > Show > Hidden items* (Figure 3)). In that

case, a folder named *.git* should appear in the current directory, which shows that the directory is now a Git repository (first folder in Figure 3).

You can now interact with this Git repository anytime by opening the directory (*mvc-self-learning-module*, not *.git*) in a terminal as before.
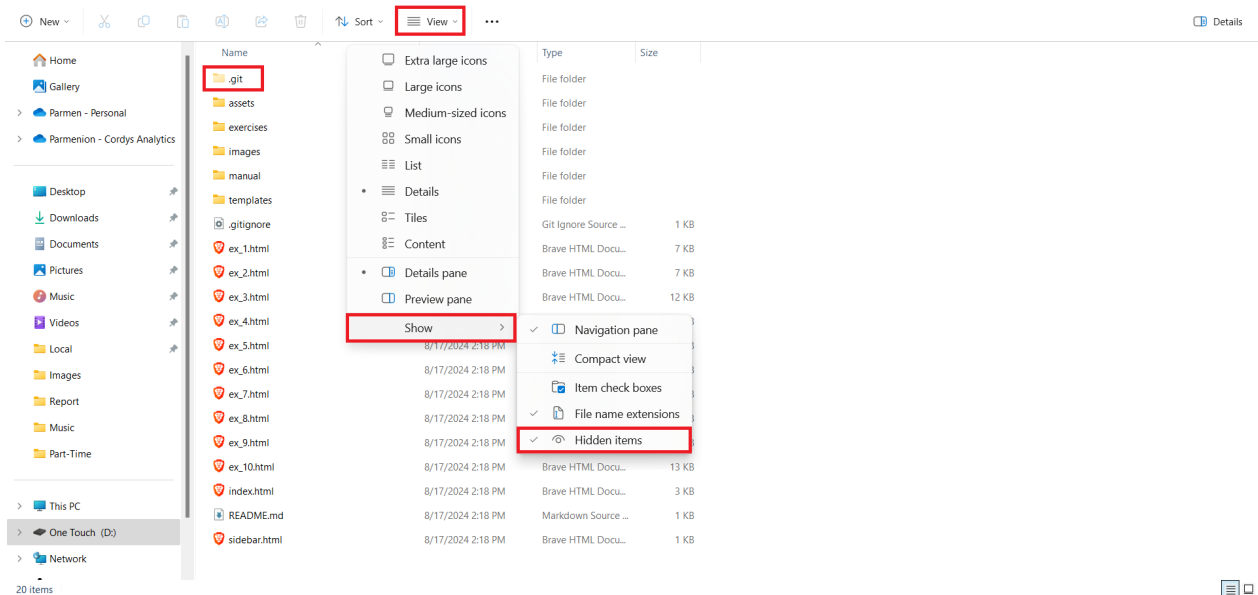

Figure 3: Enable hidden items. .git folder is shown.

Option 2: Clone an already existing Git repository from GitHub. This can be done as follows:

1. Copy the link to the repository from your browser, e.g.:

   `https://github.com/StefanMaubach/mvc-self-learning-module`

2. Navigate to the location on your computer where you would like to store the local Git repository and open a terminal within this location (*Right click > Open in Terminal*).

3. Type the command:

   `git clone https://github.com/StefanMaubach/mvc-self-learning-module`

The repository should then appear in a new directory named *mvc-self-learning-module* in the desired location.

In order to further interact with the Git repository you need to access this new directory with the terminal. You can enter the directory through the File Explorer and then open a new terminal in this directory in the same way as before, or if you still have the previous terminal open, you can access the new directory with the following command:

`cd mvc-self-learning-module`

(cd is the command used by Windows as well as Unix based systems to **c**hange **d**irectory). To verify that you cloned the repository correctly, you can check its status with the following command:
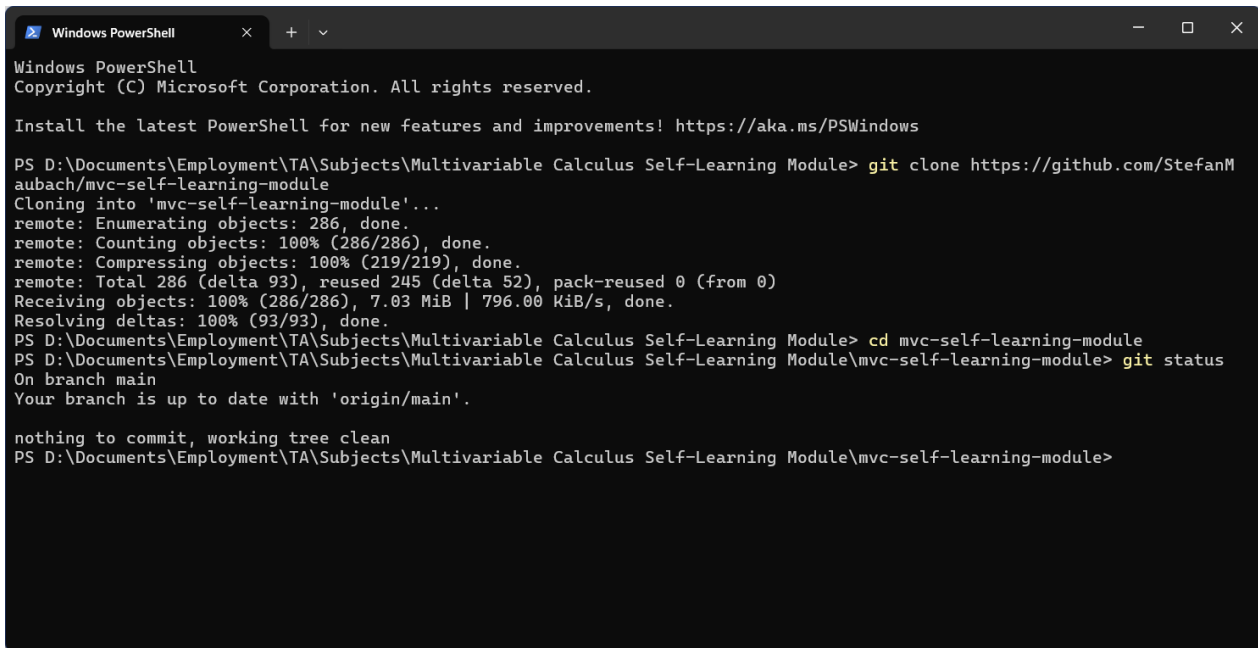
`git status`

Throughout the above process, the terminal output should look something like Figure 4.

2.1.2 Making the first commit

I now assume that you have a local Git repository. You now probably want Git to make a checkpoint out of the current state of your files by making a *commit*. This involves the following two steps:

- You tell Git what you want it to keep track of by using the command `git add`, followed by whatever filenames you want to add. In the vast majority of cases you probably want to add all of the files (except those from *.gitignore*, see Section 2.1.3) by using the dot (.) character. So the full command is:

Figure 4: Clone a GitHub repository.

```
git add .
```

You can ignore any warnings of the form "*X* will be replaced by *Y* the next time Git touches it" if they appear.

- Now it is time to finalise the commit by using the command `git commit`. Git by default asks you to label your commit by a message which briefly describes what the commit is about, e.g. "ex_1001 added" or "new sidebar layout" etc. For the first commit though it is not uncommon to choose a less descriptive message, such as "first commit" or something. The messages are added using the `-m` flag. So the full command would look something like:

```
git commit -m "first commit"
```

The terminal output should look similar to Figure 5.

Identity verification. If you have cloned a GitHub repository (Option 2 from Section 2.1.1), Git may throw an error and ask you to verify your identity before accepting the commit. The error will ask you to run the following commands:

```
git config --global user.email "you@example.com"
```

```
git config --global user.name "Your Name"
```

Run these commands, replacing `you@example.com` by the email you used to sign up with GitHub, and `Your Name` by your GitHub username (include the `"..."`). Most likely you will only have to do this once. When this is done, try committing your changes again.

2.1.3 Adding .gitignore and README.md files (optional)

If a Git repository is to be shared with others, it is a good idea for it to contain a *.gitignore* file and a *README.md* file (maybe also a *LICENSE* file if you want to clarify that it is open-source, but that is probably a future concern). It is easiest to create and edit these files locally using VSCode (see Section 2.4 for setup). Alternatively, you can create these files as follows:

1. Enable file name extensions being shown in the File Explorer by *View > Show > File name extensions* (right above *Hidden items* in Figure 3).

2. Create a blank text file in the directory by *Right click > New > Text Document*. A new text document with a name along the lines of *New Text Document.txt* should appear in the directory.

Figure 5: First commit.

3. Now you can simply replace the document's entire filename INCLUDING THE EXTENSION *.txt* by *.gitignore* or *README.md.*

You can edit these files in VSCode, or in Notepad by *Right click* (on the file) > *Open with*, and then either choosing Notepad if it appears as a suggestion, or searching for Notepad through *Choose another app.*

The .gitignore file. The .gitignore file tells Git what to not keep track of, and therefore also not to share with others. Whatever you add on .gitignore will stay in your local computer, forever invisible to Git. To add a file or even a folder to .gitignore, you simply open the file and write its name in a new row. For example, I have already included a .gitignore in the repository, which currently ignores the local LaTeX artefacts produced in my computer. Perhaps you would also like to add the entire *exercises* folder in the future so that nobody but you has access to the exercises scan/latex/pdf (although someone who is motivated enough can still view the entire exercise through the HTML files themselves).

The README.md file. The README file is what everyone expects to see as a first introduction to your repository when you share it with them. GitHub by default displays the contents of the README.md file (if it exists) to anyone who accesses your repository through their browser. I have already included a README file in the repository, which only includes a title currently. The *.md* extension means that the file is written in *Markdown*, which is an easy markup language that essentially makes fancy text files. Pretty much all the Markdown formatting you will ever need to write a typical README file can be found here:

`https://www.markdownguide.org/cheat-sheet/`

2.2 GitHub

Opening a GitHub account should be as straightforward as opening any account in any other service if you select *Sign Up* from:

`https://github.com/`

2.2.1 Associating the local Git repository with a GitHub repository

I assume that Git is already installed and that you have a GitHub account. This step needs to be done ONLY IF YOU INITIATED A NEW LOCAL GIT REPOSITORY FOR YOUR LOCAL DIRECTORY (Option 1 in Section 2.1.1). If you cloned an already existing Git repository from GitHub, you can skip this section.

Creating a new GitHub repository. If you want to create a new GitHub repository for your project, you can do that by accessing your GitHub profile through your browser, at:

`https://github.com/StefanMaubach`

and then navigating to the *Repositories* tab and selecting *New* on the right (e.g. on my profile it looks like Figure 6).
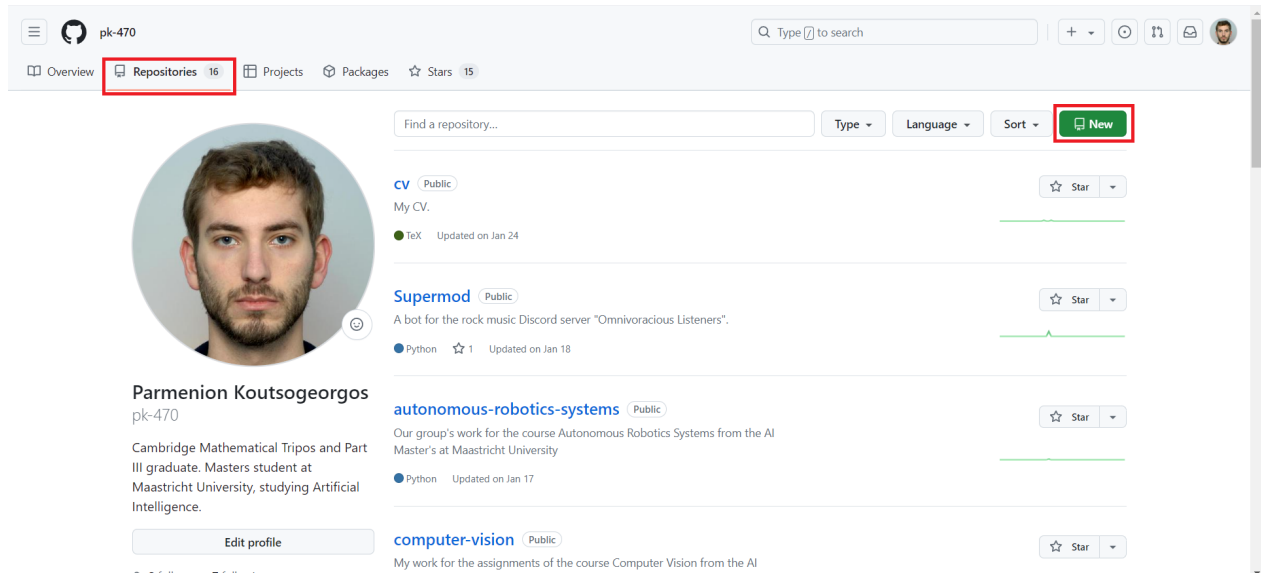

Figure 6: Start a new repository.

A menu as shown in Figure 7 should appear. The following are important:

1. Choose yourself as the repository owner.

2. Write down the local Git repository name as the GitHub repository name.

3. Make the repository public so that others will not need special permissions to view it, and more importantly so that you can deploy it on GitHub Pages for free. Note that even if the repository is public, nobody else will be able to edit it without permission.

You can also add a short description if you want. No other default options need to be changed. You can click *Create repository* after you are done. You should now be redirected to your new repository, at:

`https://github.com/StefanMaubach/mvc-self-learning-module`

Associating your local Git repository with an empty GitHub repository. I assume now that:

- You have a local Git repository for your project, preferably (but not necessarily) one at which at least one commit has been made.

- You have an empty GitHub repository in which you want to push your local Git repository.

If the GitHub repository is indeed still empty, a helpful guide such as in Figure 8 should appear when you access the repository through a browser. Make sure to select the *HTTPS* option of the guide as the SSH option requires further setup (although nowadays the SSH option is often preferred because of security reasons). Assuming that you already have a local Git repository, simply open its directory in a terminal as before and follow the second block of instructions shown in the guide. A few words about each line:

1. The first line:

   `git remote add origin https://github.com/StefanMaubach/mvc-self-learning-module.git`

   is the one that actually associates the local Git repository with the remote GitHub repository.

2. Technically the second line:

   `git branch -M main`

Figure 7: New repository options.

is not needed. It is there because the main Git branch used to be called *master* instead of *main*, so this line is there to simply ensure compatibility with older repositories. If you installed Git using the default options, you do not need this.

3. The third line:

```
git push -u origin main
```

is needed ONLY IF YOU HAVE MADE AT LEAST ONE COMMIT TO YOUR LOCAL GIT REPOS-ITORY. Otherwise this line will throw an error in the terminal (which you can ignore), as there is no commit to push. You may skip this line until you are happy with a version of your website that you want to be up and running. Whenever you want your files to be pushed to GitHub however, so that the website can be deployed, go back to Section 2.1.2 and make your first commit, and then use this line to push it to the main branch on the remote GitHub repository.

The terminal output should look something like Figure 9.

Figure 8: New repository setup.

2.3   GitHub Pages

It is time to deploy the website on GitHub Pages! I now assume that:

- You have a local Git repository for your project at which at least one commit has been made.
- You have a GitHub repository associated with your local Git repository.
- You have pushed your local commits to the remote GitHub repository.

If you access the GitHub repository through your browser, it should look something like Figure 10.

In order to deploy the repository as a website on GitHub pages, go to *Settings* (on the top) > *Pages* (on the left). Then at section *Build and deployment* and subsection *Branch* choose the *main* branch and click *Save* (Figure 11).

2.3.1   Link to the website

If you go back to your main GitHub repository page after deployment, you should see a new section named *Deployments* on the right sidebar (Figure 12).

Clicking on *github-pages* should lead to a page similar to Figure 13.

Figure 9: Associate local Git repository with remote GitHub repository and push the local commits.
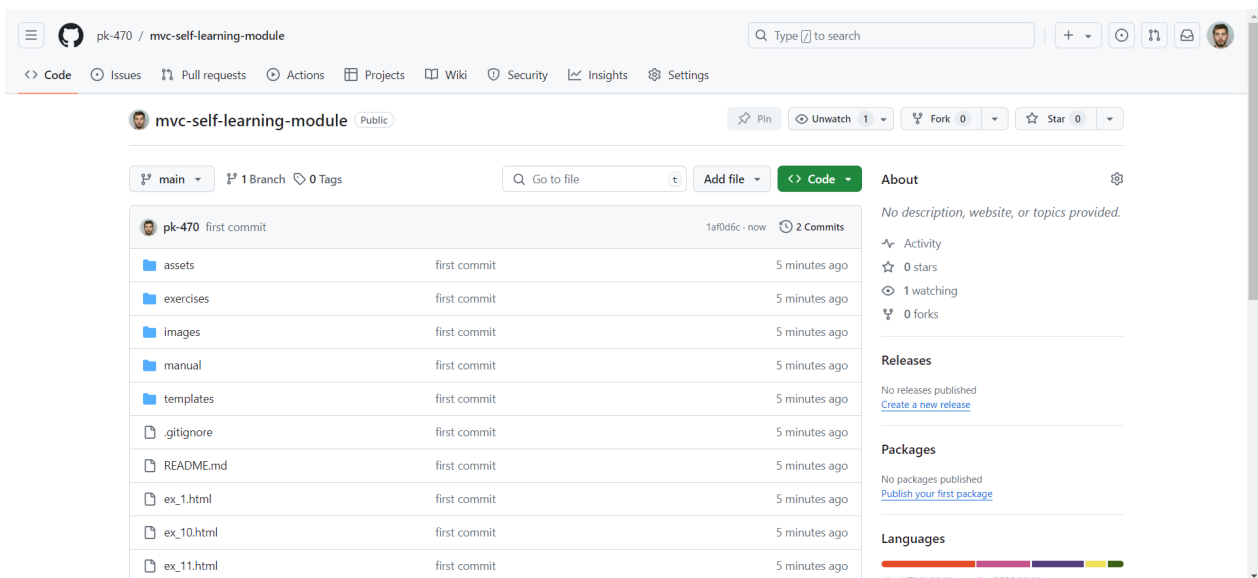
Figure 10: GitHub repository after setup and first push.
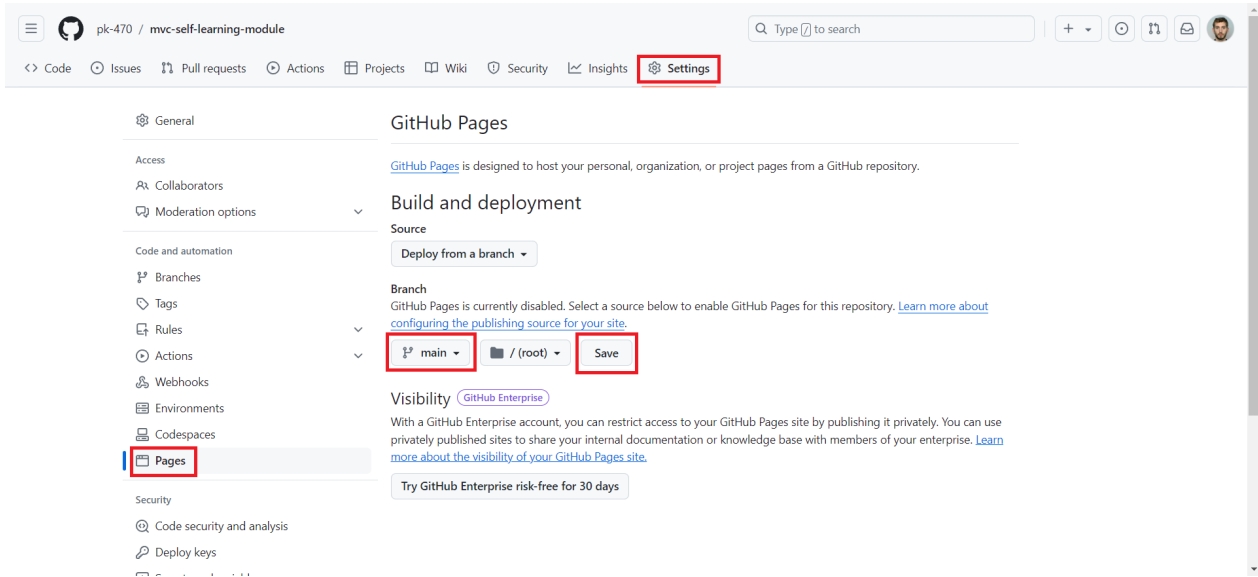
Figure 11: Deploy the repository on GitHub Pages.
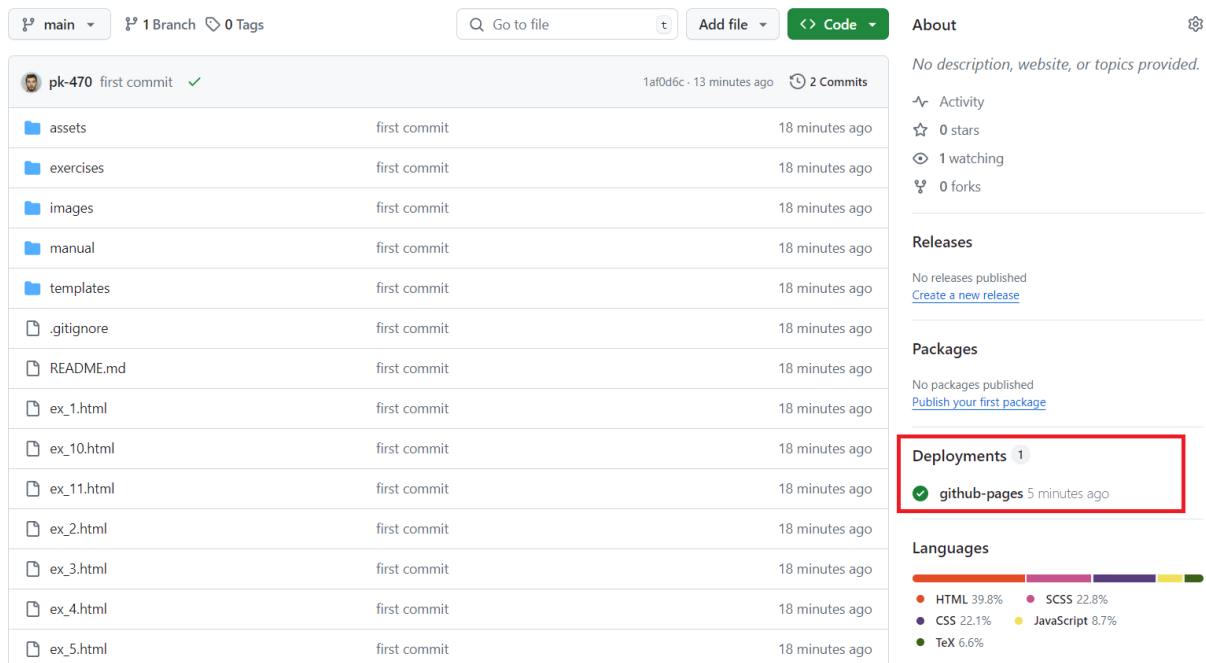
Figure 12: Deployment section added to the sidebar of the repository.
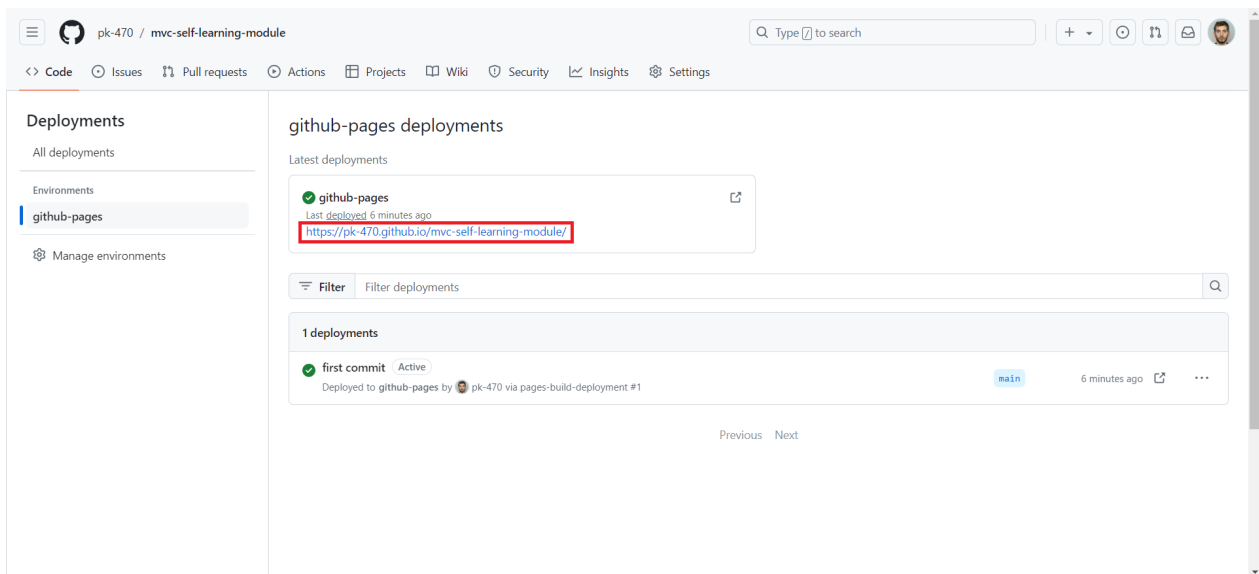
Figure 13: Deployment page for the repository.

Finally, following the first link should lead to the website (Figure 14). The website will remain in this link (unless you decide to change the domain, which can be done but is more complicated), and will update itself whenever you make a new commit and push it to the GitHub repository.
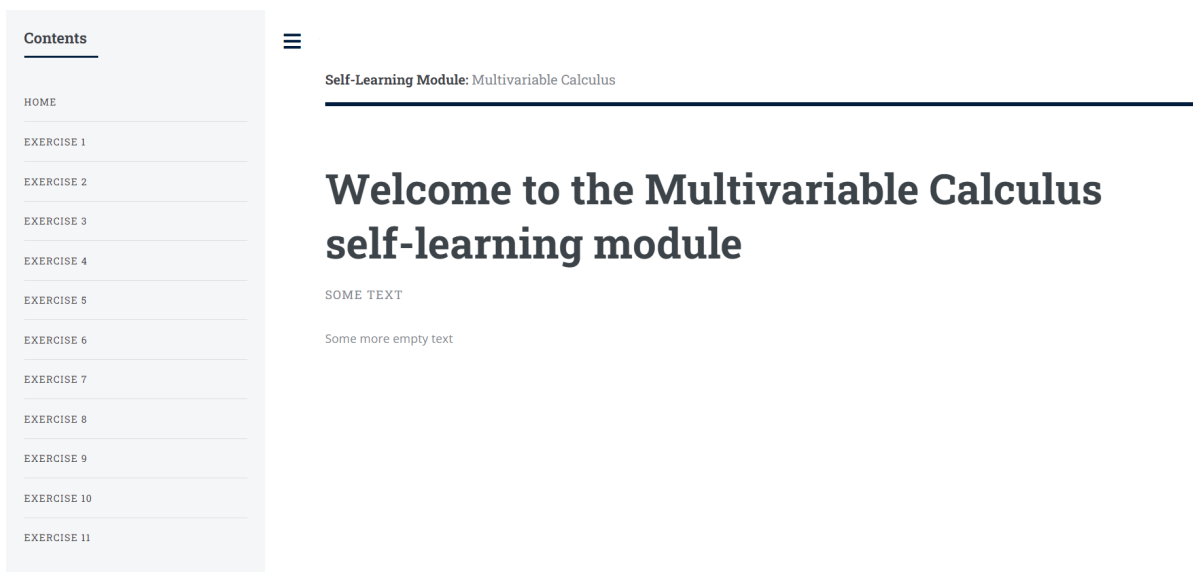
Figure 14: Deployed live website.

2.4 VSCode (optional)

I recommend using Visual Studio Code (VSCode) to edit HTML or any other files that you don't already know how to edit easily. VSCode is a free and lightweight text editor that can be customised to work with almost every programming or markup language (including LaTeX) using extensions. You can download VSCode from:

`https://code.visualstudio.com/`

The default installation options are fine.

In order to add extensions to VSCode, you need to click on the block-looking icon on the left sidebar. This should open a small window next to the left sidebar, which is titled *Extensions* and contains (e.g. Figure 15):

- a search bar;

- a list with your installed extensions;

- possibly, some recommended extensions.
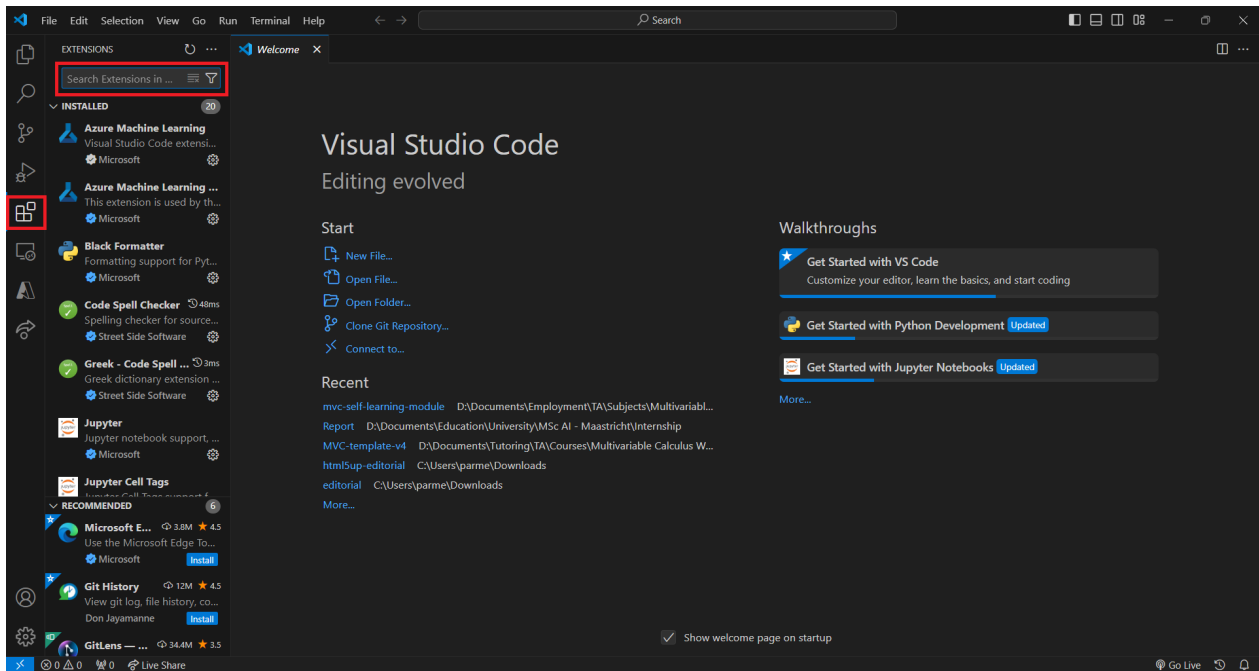
Figure 15: VSCode extensions.

You can use the search bar to search for new extensions to install. For the purposes of this project I recommend the following two extensions:

Live Server. This extension is used to produce a local live server so that you can run and directly see the changes you make while editing the website. It is also useful because some JavaScript components will not run when you view the website through the Windows file system, i.e. when the path on your browser looks like *C://...* instead of *http://...* or *https://...*. You can install *Live Server* as shown in Figure 16. If you have not already installed the extension, there should be an *Install* option around where *Uninstall* is in the picture.

Prettier. This extension is used to format code in a way that makes it easier to read, which makes bugs easier to see, which has been proven to significantly increase life expectancy over time. You can install *Prettier* as shown in Figure 17. If you have not already installed the extension, there should be an *Install* option around where *Uninstall* is in the picture.

2.5 Python

You can download the latest version of Python from:

`https://www.python.org/downloads/`

The website should recommend the latest installer for Windows. During the installation process, MAKE SURE TO CHECK THE BOX THAT ADDS PYTHON TO PATH! Otherwise, you will have to do it manually in order to run Python scripts. In order to do it manually, Google something like "add python to path" followed by your system, and follow the detailed instructions.
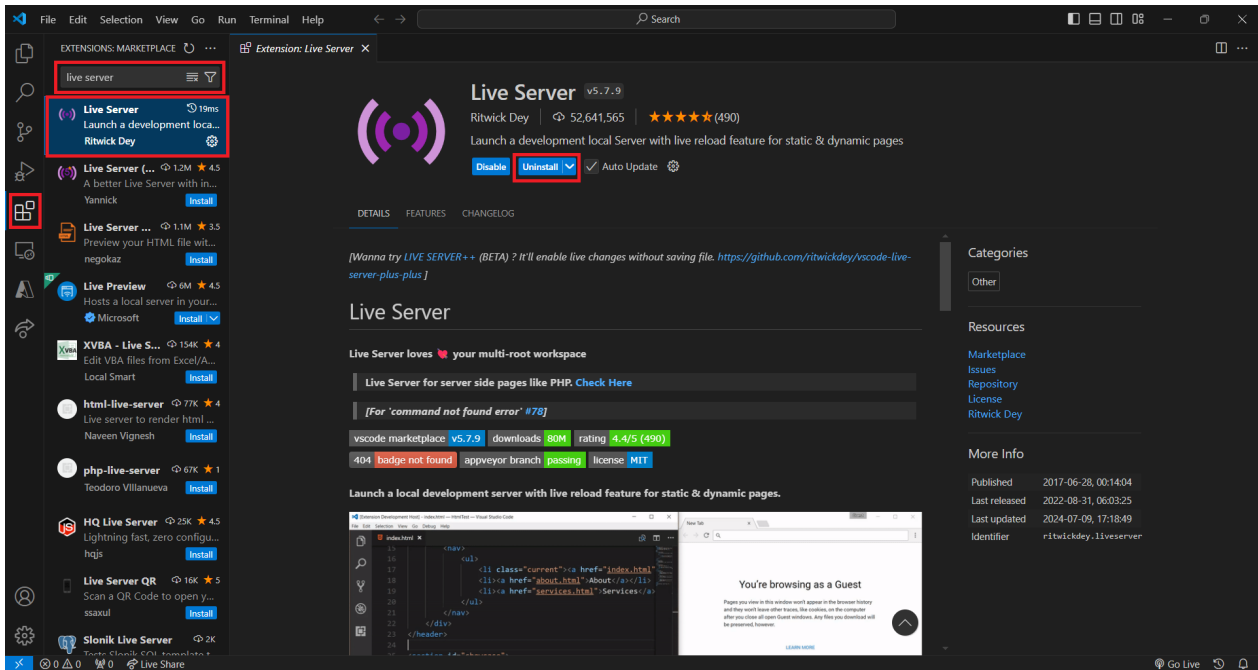
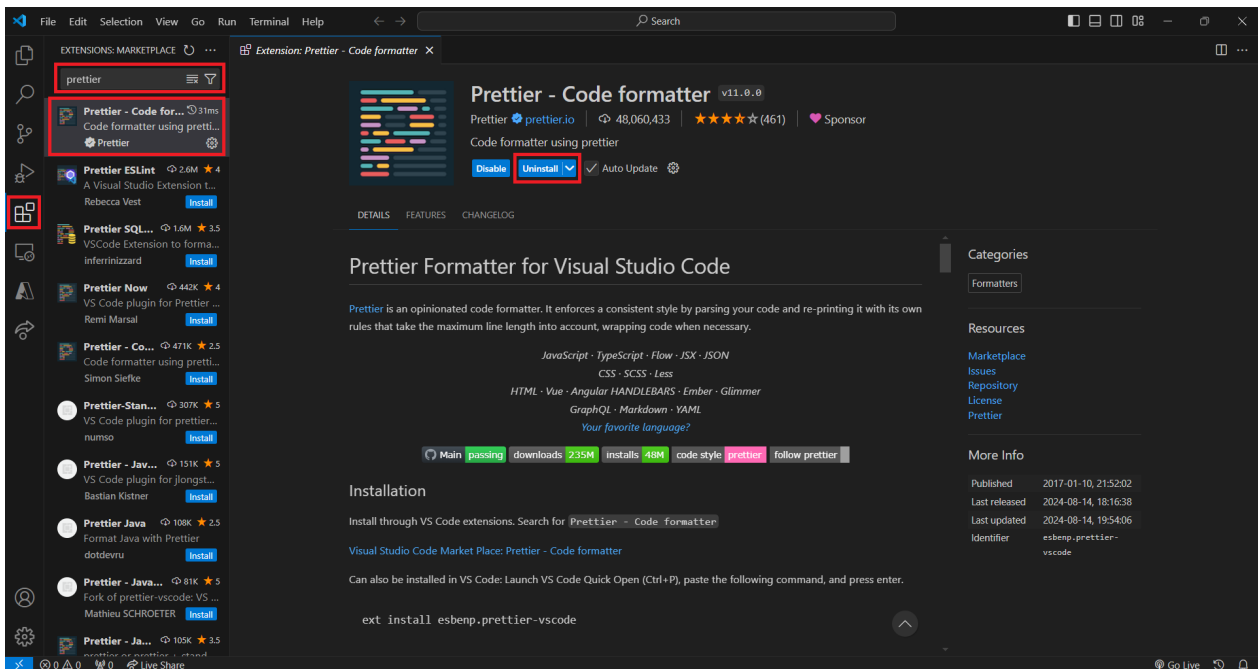Figure 16: *Live Server* extension.

Figure 17: *Prettier* extension.

3   Usage

Finally, everything is set up! You should now have:

- a local Git repository for your project at which at least one commit has been made;
- a GitHub repository which is up to date with your local Git repository;
- an up and running website deployed on GitHub Pages from your GitHub repository.

From now on, things should be much easier!

3.1   Updating the website

In order to work with the update pipeline (Git $\rightarrow$ GitHub $\rightarrow$ GitHub Pages), you only need the following steps:

1. Pull any changes made to the repository from someone else. This should be done IF AND ONLY IF THERE ARE MULTIPLE COMPUTERS WORKING ON THE SAME BRANCH. Otherwise, Git may find conflicts between the changes made in each computer's local commits and those in the global repository, and you may need to enter the super annoying process of merging conflicts, i.e. telling Git what to keep from each commit. If you are only working on the branch on a single computer yourself, feel free to skip this step. If you need to perform it however (e.g. if you work on the repository from multiple computers, or if you get another TA to work on this and they don't use a separate branch), use the command:

   `git pull origin main`

   IMPORTANT NOTE: After setup is complete, always use `git pull` and never `git clone`. Otherwise you will clone the repository as it is on GitHub again, and you will lose any local changes!

2. From now on, you simply make your changes, and commit in the exact same way as the first commit. For example, I made some changes in this manual. I will commit them as follows:

   `get add .`

   `git commit -m "manual ctd"`

3. Finally, push the changes to the GitHub repository. Just like in the first push, you can do this as follows:

   `git push origin main`

A couple of notes:

- During the first or third steps above, Git may require from you to verify your identity by opening a popup window and asking for your GitHub credentials. Simply login as it says, and ignore or close any extra browser windows that may appear.
- If you included the `-u` flag in the first `git push`, technically you can use `git pull` and `git push` without any arguments. However, I think consistency and clarity makes things easier.

That's it! GitHub Pages will now see the new changes from your GitHub repository and it will automatically deploy the new version of the website along with the latest edits.

If you are using VSCode to edit your files, you can even open the working directory in a terminal within VSCode as shown in Figure 18 and type the above commands there. The output should look something like Figure 19.

3.1.1   Using the Python script to convert LaTeX to HTML

3.2   Working with VSCode (optional)

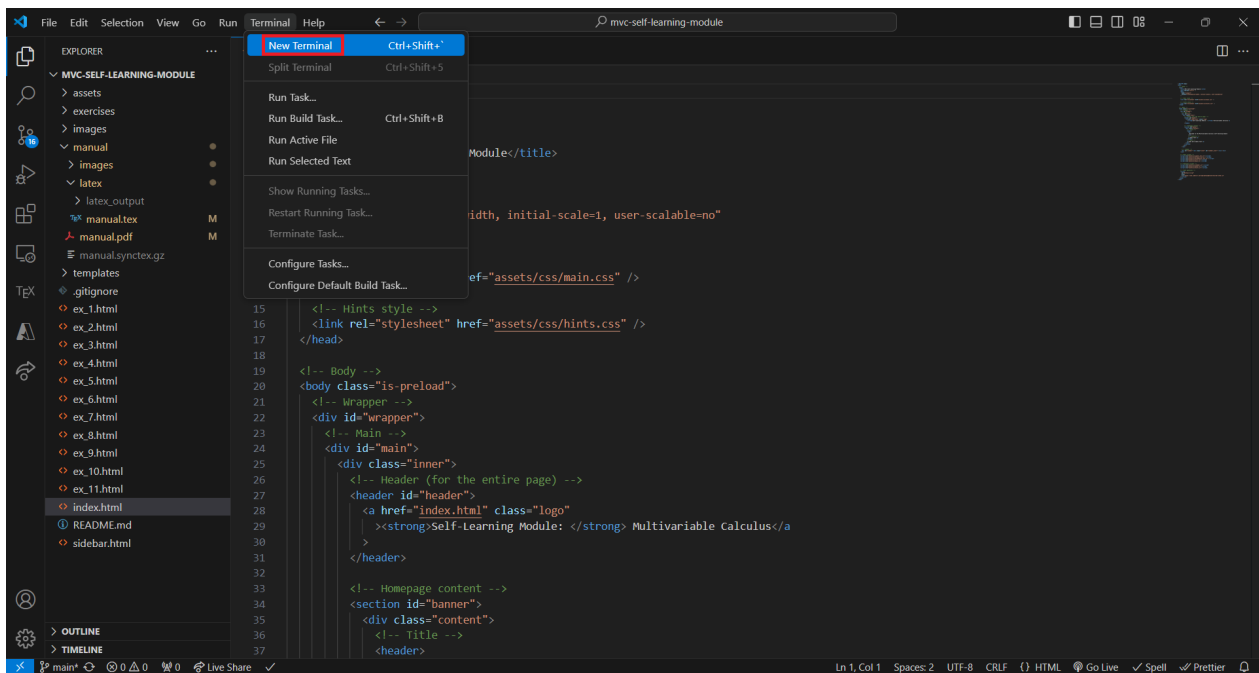Here are some general instructions about working with VSCode.
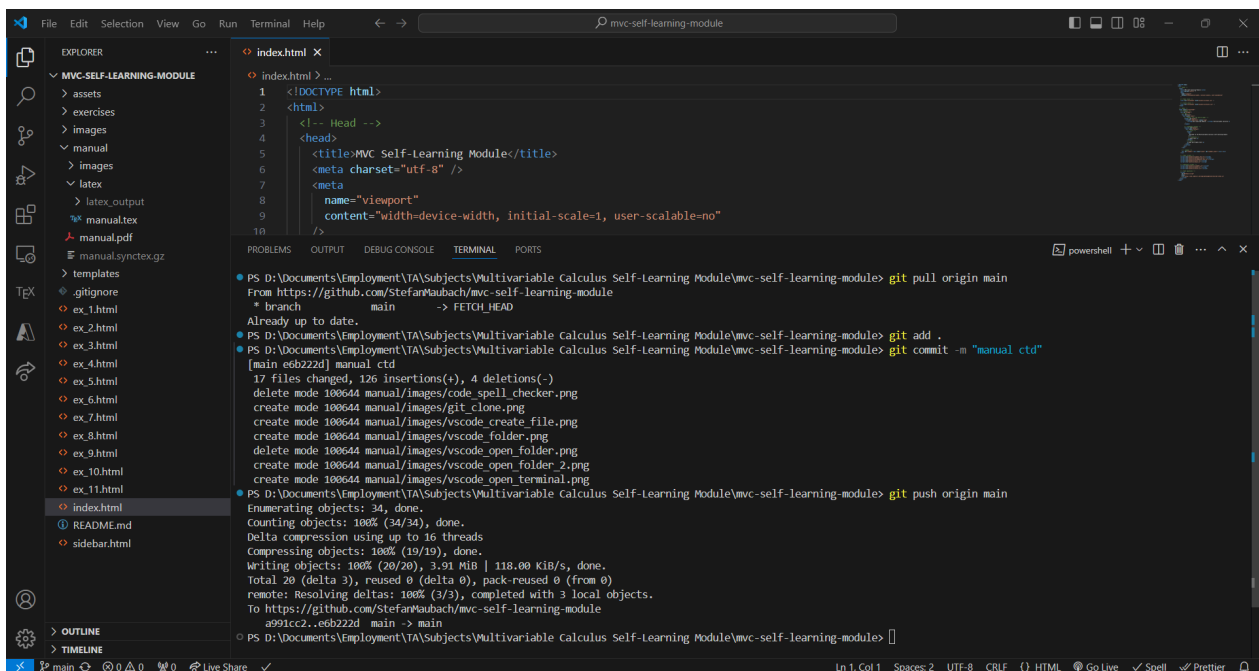
Figure 18: Open a terminal in VSCode.

Figure 19: Working with Git from a terminal in VSCode.

3.2.1 Open a folder in VSCode

There are multiple ways to open a folder in VSCode. If you have opened a new window without any open folders in it, you can:

- Choose the *Open Folder* option from the main window, and search for your folder through the popup File Explorer.

- Click on the file-looking icon on the left sidebar, choose the *Open Folder* option from the *Explorer* window that appears on the left, and search for your folder through the popup File Explorer.

- if you have opened the desired folder before, you may be able to see your folder in the *Recent* section of the main window and open it directly.
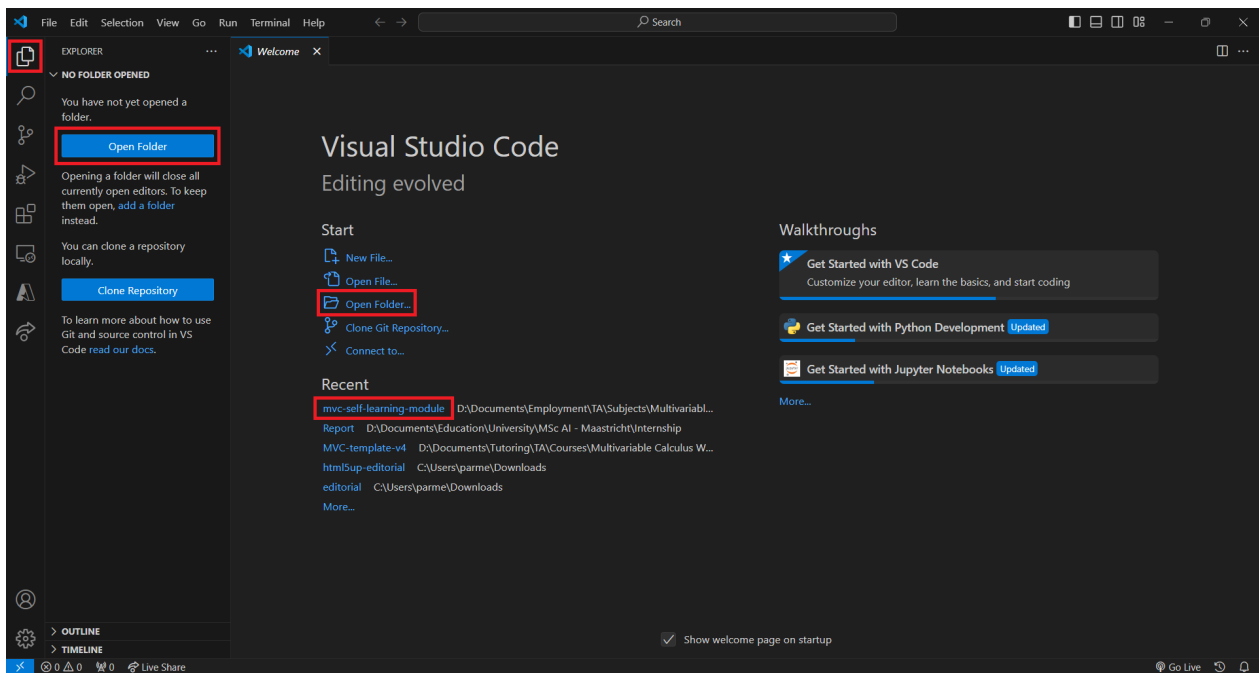
All of these options are highlighted in fig. 20.

Figure 20: Open a folder in an empty window in VSCode.

If you already have a folder open and you want to open another folder in a new window, you can click on *File* on the top menu, choose *New Window* from the dropdown menu, and choose any of the previous options in the new empty window that will appear (e.g. Figure 21).

Finally, if you want a backup option that will always work without opening any new windows, you can click on *File* on the top menu, choose *Open Folder* from the dropdown menu, and search for your folder through the popup File Explorer (e.g. Figure 21).

3.2.2 Create a new file within an open folder in VSCode

I now assume that you have opened a folder in VSCode. In order to see the contents of the folder, click on the file-looking icon on the left sidebar. In order to create a new file in this folder, right click on any empty space on the *Explorer* window that appears on the left, choose *New File. . .*, and write the full name of the file you want to create INCLUDING THE EXTENSION NAME (e.g. *ex_1001.html* in full, e.g. Figure 22).

3.2.3 Edit a file from an open folder in VSCode

In order to open a file for editing in VSCode, click on the file-looking icon on the left sidebar to open the *Explorer* window, find your file, and DOUBLE CLICK on it to open it for editing. If you click only once, the file will be open for viewing, so if you open a different file immediately afterwards, the previous file will be gone from the main window. The purpose of this is to avoid clutter when you're quickly shuffling through
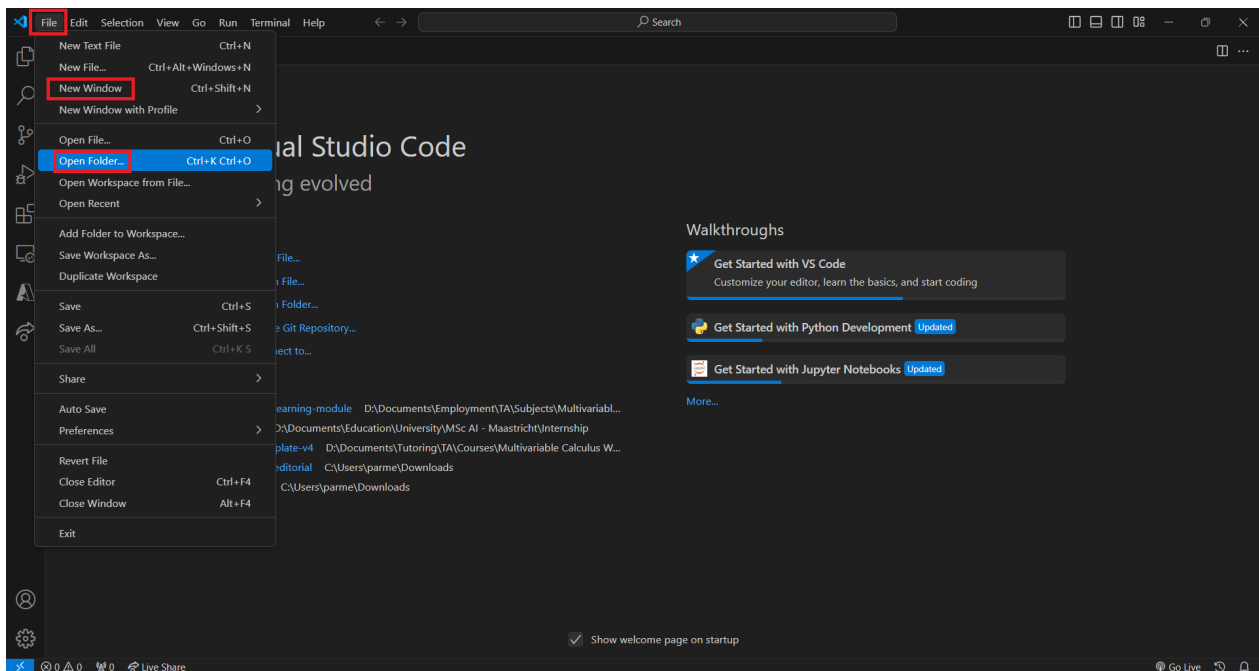
18

Figure 21: Open a new window in VSCode, or open a folder through the *File* menu.
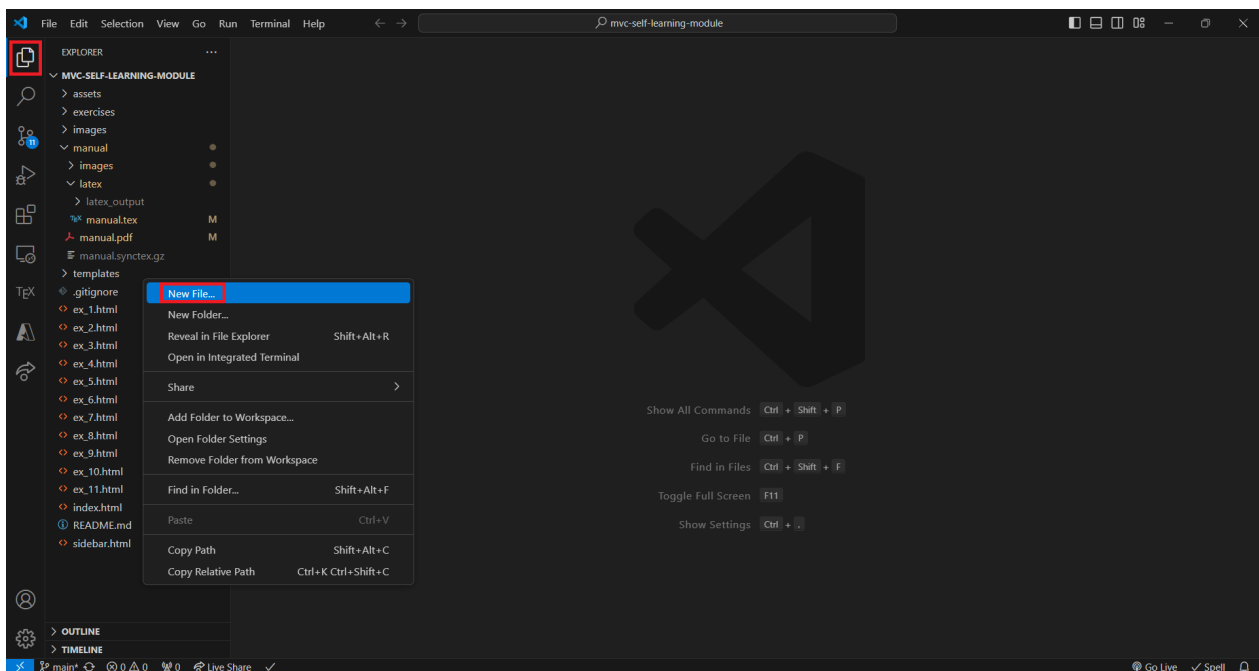

Figure 22: Create a file within an open folder VSCode.

files while looking for something. Having said that, even if you open a file by clicking only once and then you start editing it before opening another file, that also works.

3.2.4 Format a file in VSCode

In order to set up formatting options for a file in VSCode, open the file for editing, and then *Right click > Format Document With...* (e.g. Figure 23). A popup menu should appear on the top of the main window. If you haven't chosen a formatter before, you can click *Configure Default Formatter...* at the bottom of this window, and then choose *Prettier* from the list that will appear (e.g. Figure 24). From that point on, you can format your document anytime by *Right click > Format Document* (or by using the shortcut *Shift + Alt + F* if you are willing to remember it).
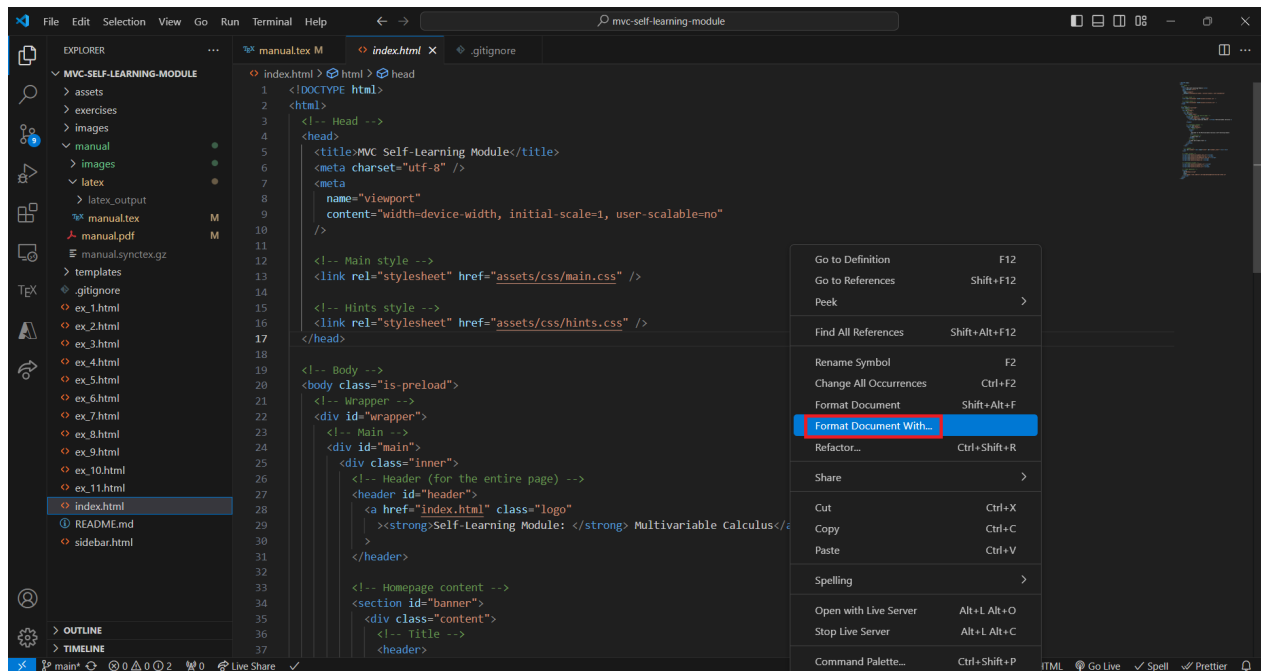

Figure 23: *Format Document With...*.

Format on save. I suggest that you set up formatting to happen automatically on save (*Ctrl + S*). It is quicker this way, and even if you accidentally save before you're done and then formatting messes up what you were working on, you can always revert back to the previous state of the document by *Ctrl + Z*, aka *Undo*. To enable format on save, follow the steps:

1. Open VSCode settings by *File* (on the top menu) > *Preferences* > *Settings* (e.g. Figure 25).

2. Click on the *Formatting* bookmark in the main window, scroll down just a bit, and click on *Format On Save* (e.g. Figure 26).

You don't need to do anything else to save the settings, the changes should be effective immediately.

3.2.5 Using Live Server

If you are using VSCode to edit your HTML files, you can run the website on a local live server using the *Live Server* extension (see Section 2.4). All you need to do is click on the *Go Live* option on the bottom menu of VSCode (e.g. Figure 27). The local version of the website should open in your default browser in the url (e.g. Figure 28):
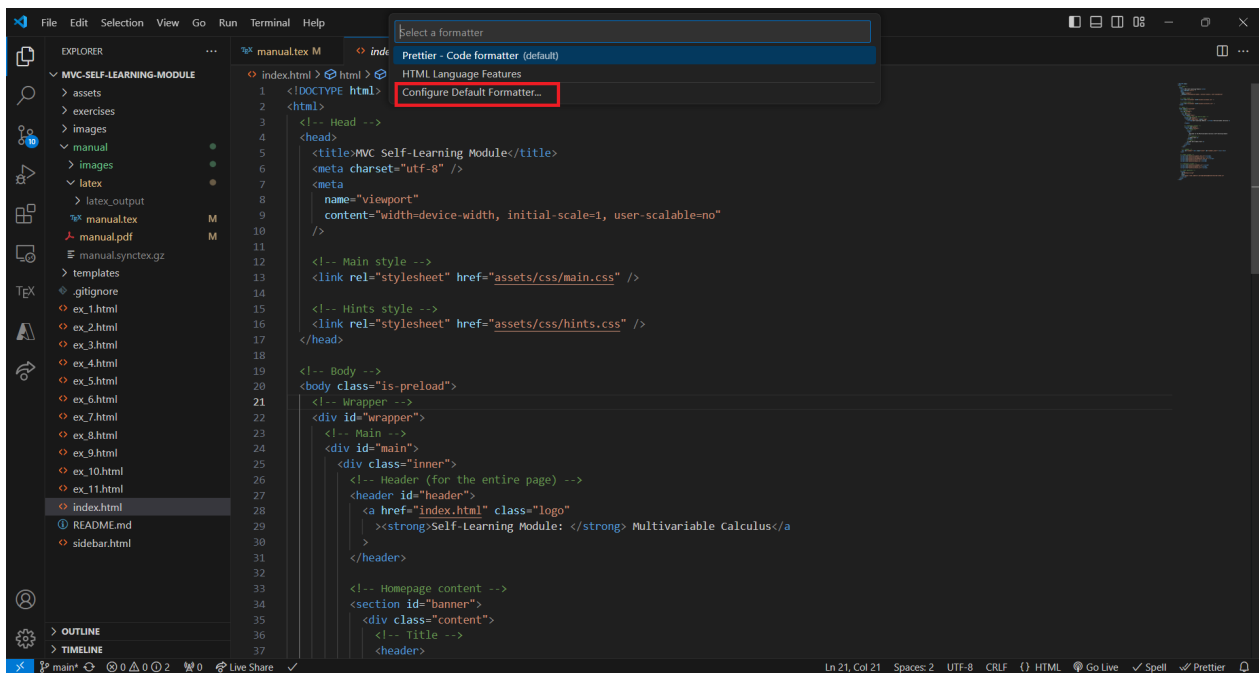
```
http://127.0.0.1:5500/
```
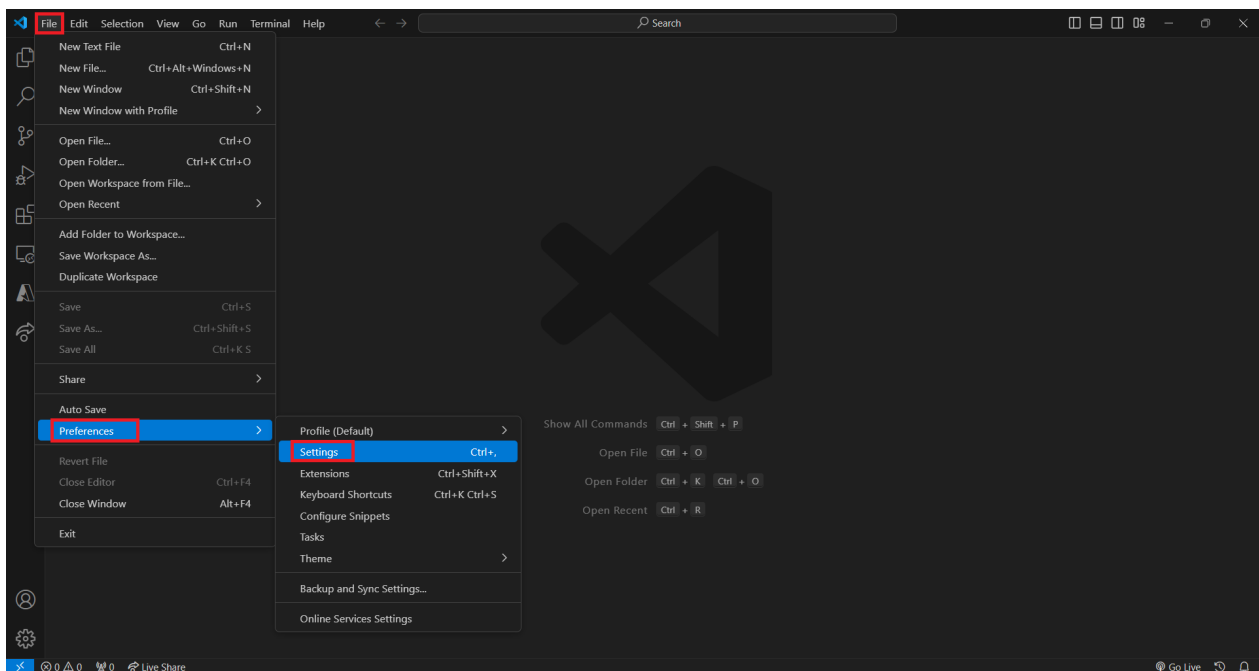
Figure 24: *Configure Default Formatter....*

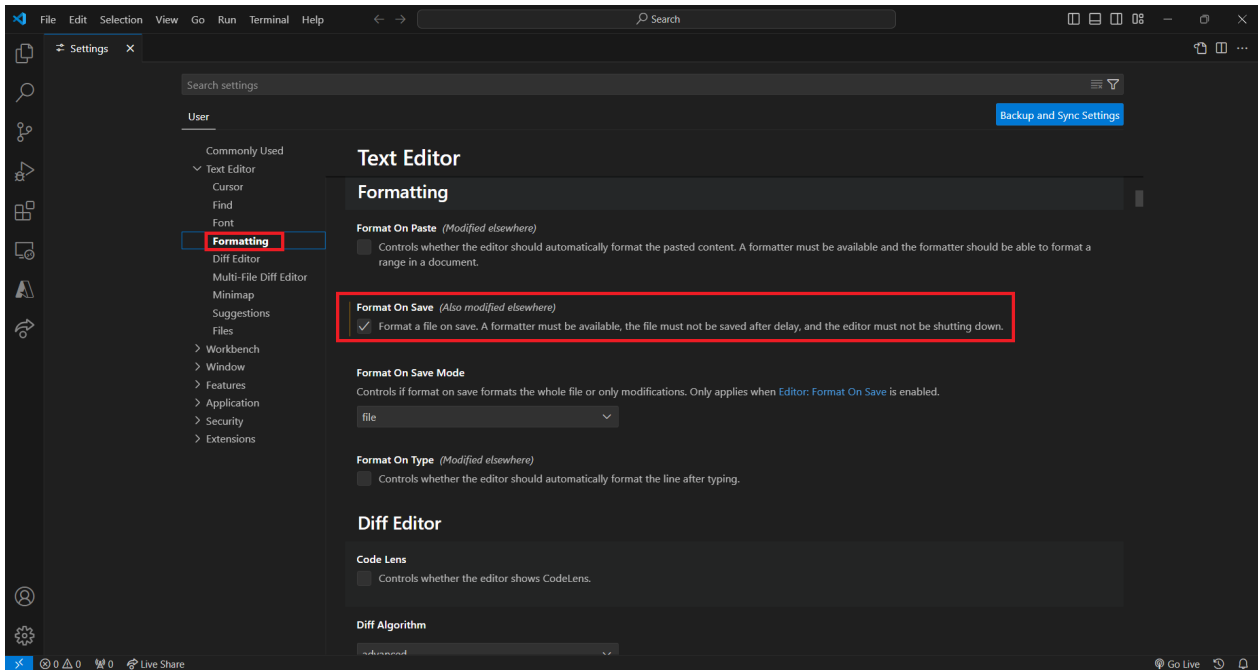Figure 25: Open VSCode settings.

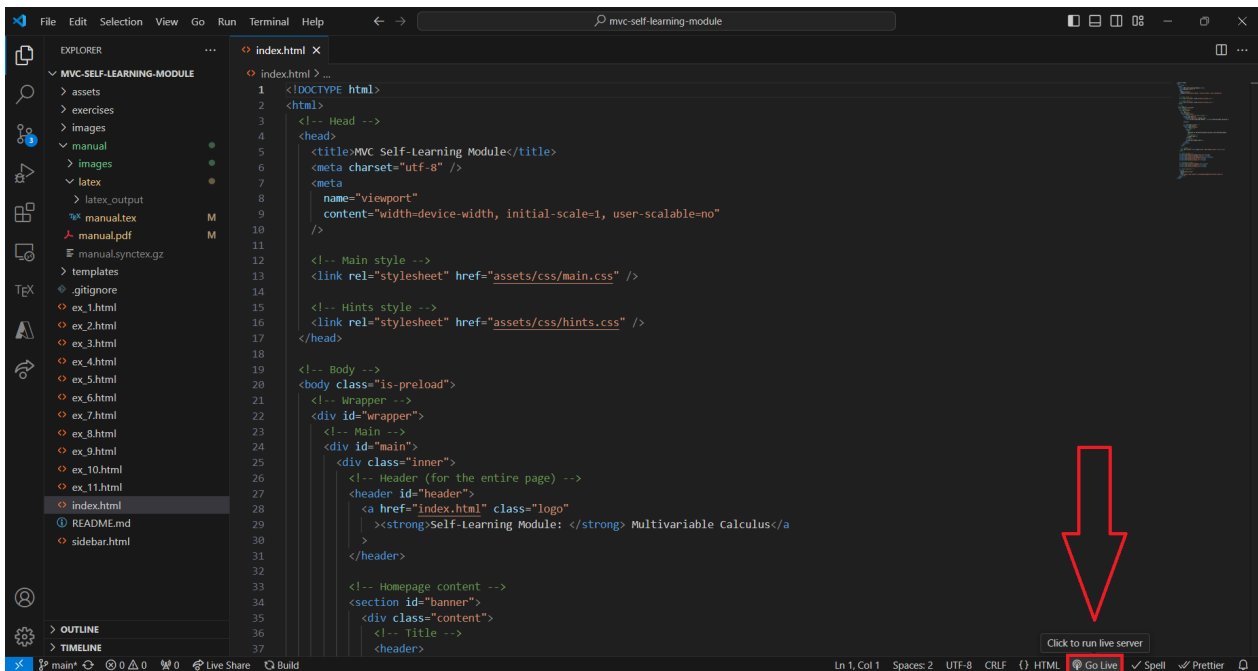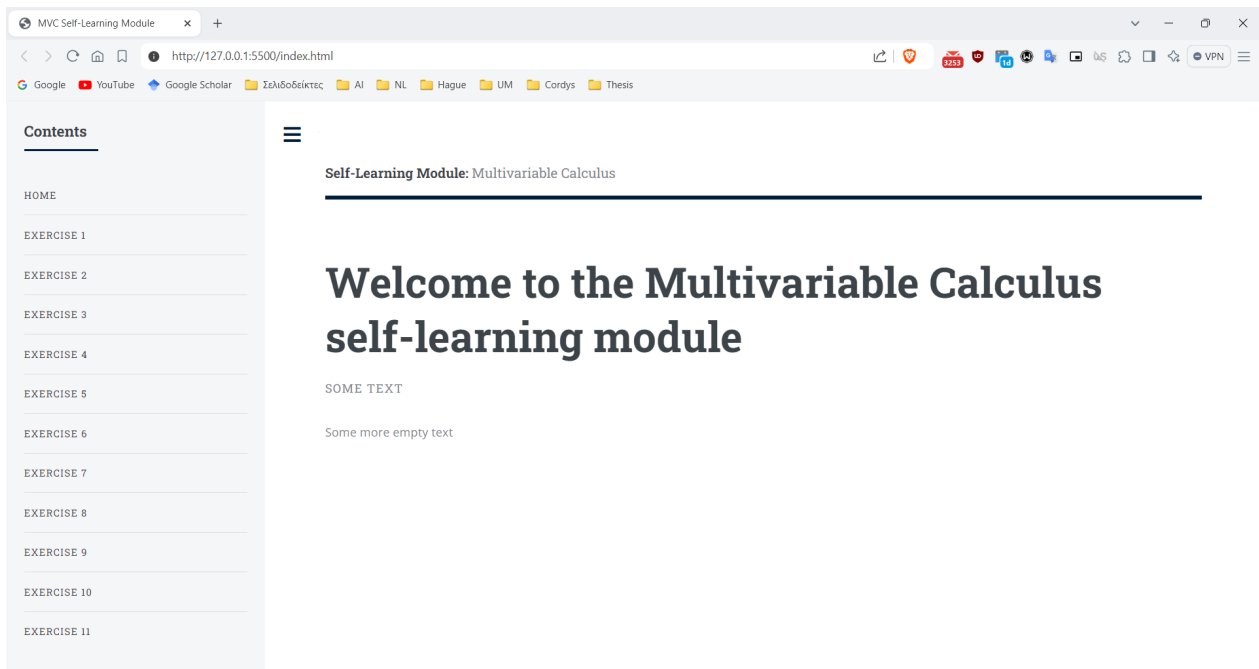Figure 26: Enable format on save.

Figure 27: Opening *Live Server* in VSCode.

Figure 28: Website running on local live server.