|  |  |
| --- | --- |
| CL1002  Programming Fundamentals Lab | Lab 02  Introduction to Pseudocode, Algorithms and GitHub |

# National University of Computer and Emerging Sciences

Fall 2024

# Aims and Objectives

**Aims:**

* Enable students to effectively plan and outline programs using pseudocode.
* Provide students with the skills to translate logical steps into structured algorithms.
* Familiarize students with essential programming keywords and operations.
* Introduce students to the basics of GitHub for managing and sharing code.

**Objectives:**

* Understand the concept of pseudocode and its role in program design.
* Learn the specific keywords and operations used in pseudocode.
* Develop the ability to write algorithms that outline the logic of a program.
* Gain a basic understanding of GitHub as a version control and collaboration tool.

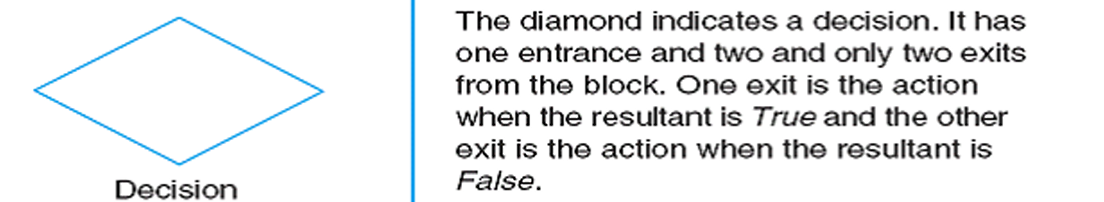
# Introduction

In this lab we will introduce the following concepts.

Flowcharts (Continued from previous Lab)  
Pseudocode   
Algorithm  
GitHub

# Flowcharts (Cont.)

In the previous lab we discovered what flowcharts are and why they are useful for programming applications. But flowcharts are not just input, process and outputs. They contain many more blocks that define multiple programming structures.

A close-up of a white background

Description automatically generated

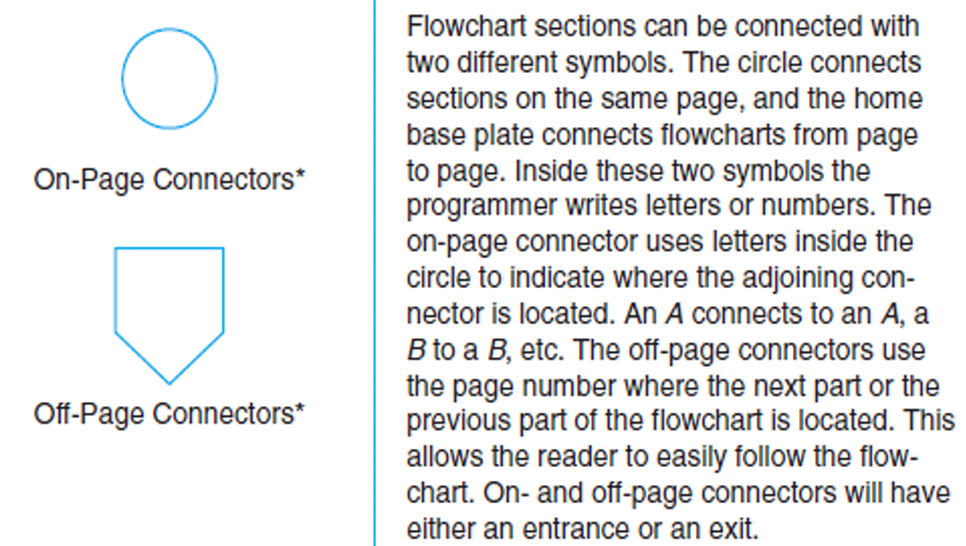
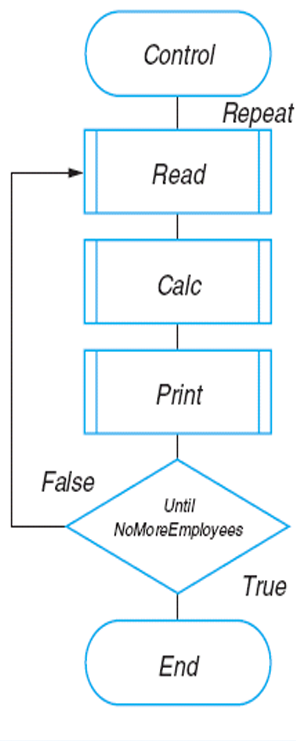


Figure 1. Different shapes for flowchart design

# Example

Let’s look at the pay roll example from our previous lab.

A white rectangle with black text

Description automatically generatedA white rectangle with black text

Description automatically generatedA white rectangle with black text

Description automatically generated

Figure 2. Example of Payroll using the process module

The one flowchart on the left is the actual flowchart with three process modules to separate the different processes from the flowchart. The right three flowcharts are the result of the three separate process modules and are useful if the flowchart is getting bigger in size. A similar strategy is used in programming where individual blocks of codes are connected to a single driver code instead of a big file which contains all the codes.

Some Problems to Solve

1. You are working at Toyota Indus Motors and want to assemble a car. Design a flowchart with proper process modules and decision structures to replicate a pipeline production.

# Pseudocode

Pseudocode is a way to express an algorithm or program logic in a human-readable form, using plain language and simple notations that resemble programming constructs. It serves as an intermediate step between the problem statement and the final implementation in a programming language. Pseudocode does not follow strict syntax rules like a programming language, but it uses common structures such as loops, conditionals, and function calls to outline the logic clearly.

Pseudocode Conventions:

* Start/End: Indicate where the pseudocode begins and ends.
* Input/Output: Specify inputs and outputs.
* Process Steps: Use plain language to describe actions, e.g., SET, ADD, SUBTRACT, PRINT.
* Conditionals: Use IF, THEN, ELSE, ENDIF.

# Example

1. START
2. // Input/Output
3. INPUT number1
4. INPUT number2
5. // variables and Initialization
6. SET sum to 0
7. // Process Steps
8. SET sum to number1 + number2
9. // Conditional Statements
10. IF sum > 0 THEN
11. PRINT "The sum is positive"
12. ELSE
13. PRINT "The sum is non-positive"
14. END

In this example we are taking two integer numbers as input and adding them to a variable sum. After addition a decision is made if the value in the sum variable is greater than zero it returns a print statement that the variable is positive otherwise the variable is negative.

Some problems to solve.

* Find the maximum number in any of three variables.
* Take three variables as input and add them without using the + operator (Use your head for this)
* Create a small calculator which only does ‘+’ or ‘-‘Operations. (Hint: Take three variable inputs with one being used for the operator)

# Algorithm

An algorithm is a finite set of instructions that, if followed, accomplishes a particular task.

**OR**

An algorithm is a set of obvious, logical, and sequential steps that solve a specific problem.

*Characteristics of a Good Algorithm:*

* Clear and Unambiguous: Each step is precisely defined.
* Finite: It has a definite end.
* Efficient: It accomplishes the task using minimal resources.

# Example

Write an algorithm that can calculate the area of a rectangle. The width and the height of the rectangle should be taken from the user.

*Note: Area = Width × Height*

*Solution A – Good:*

1. Ask the user to enter **Width**

2. Ask the user to enter **Height**

3. Set **Area** to **(Width × Height)**

4. Display Area for the user

 • **You can describe the steps in your way, but your description of the steps should be obvious, logical, and sequential.**

*Solution B - Bad:*

1. Ask the user to enter **Width**

2. Ask the user to enter **Height**

3. Calculate **Area**

4. Display Area for the user

**The reason for considering Solution B as a bad solution:**Step 3 is not clear because it does not explain how we can calculate Area. So, this algorithm is bad because its steps are not obvious.

*Solution C - Bad:*

1. Set Area to (**Width × Height**)

2. Ask the user to enter **Width**

3. Ask the user to enter **Height**

4. Display Area for the user   
**The reasons for considering Solution C as a bad solution:**

We don't know what Width and Height at Step 1 are. In other words, Width and Height have not been defined before Step 1, so we cannot use them because they do not exist yet.

What about Step 2 and Step 3? Width and Height are defined there! After Step 2, Width does exist, but Height does not. After Step 3, Height does exist. Both Width and Height are available to be used at or after step 4.

So, this algorithm is bad because its steps are not correctly sequential.

*Solution D - Bad:*

1. Set Area to (**Width × Height**)

2. Display Area for the user  
**The reasons for considering Solution D as a bad solution:**

• Step 1 tells us to multiply Width and Height, but we don't know what Width and Height are. Even, they have not been defined in any steps of the algorithm.

• So, this algorithm is bad because of the illogical step, which is using unknown things (Width and Height).

Some Problems to Solve:

* Implement an algorithm for determining if an Nth is a divisor of an n Number (i.e. 2 is a divisor of 6). If so, determine if it’s an even number or odd number as well.
* Implement an algorithm where the user enters a number, and an appropriate month is displayed.
* Implement an algorithm for making a simple calculator with all the operators (+,-,\*,/,%)

# GitHub

GitHub is a web-based platform that provides version control using Git, allowing multiple people to collaborate on projects, track changes, and manage code repositories efficiently. It’s widely used by developers for both open-source and private projects.

# Create an account on GitHub

To get started first login to GitHub (if you already have an account).

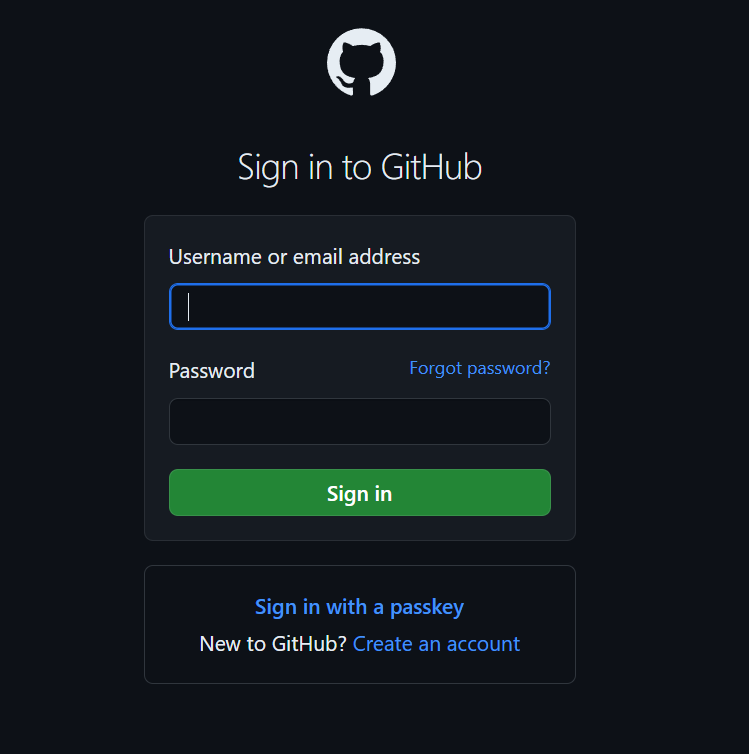


Figure 3. GitHub Sign-in Page

If not, then click on the **create an account** option and enter the necessary credentials to get started. Once logged in you should see a screen something like.

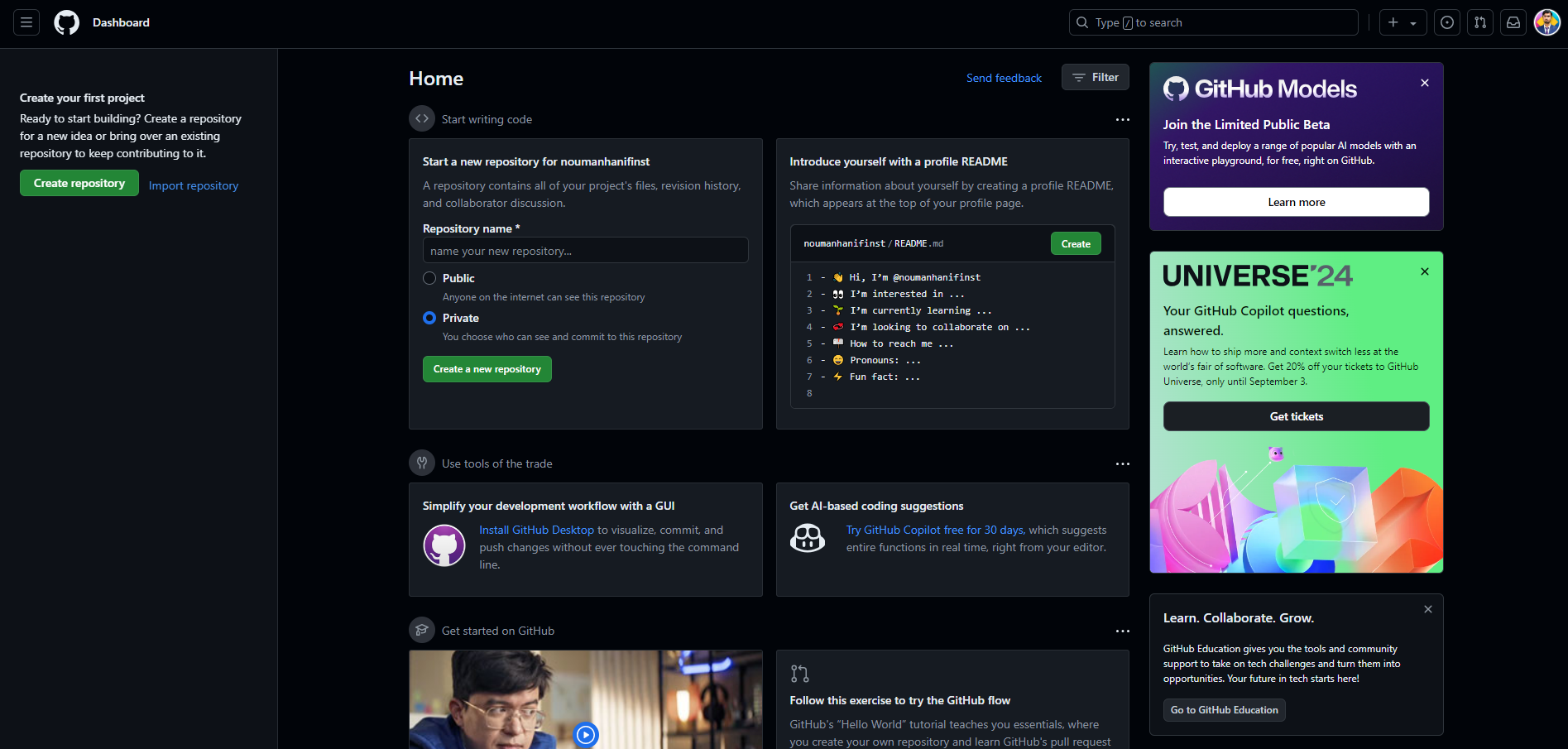


Figure 4. GitHub Dashboard

Of course, mine has many repositories so when performing your screen would look a bit different. On the top left section where you see the new button, click on it and you should see a window something like this.

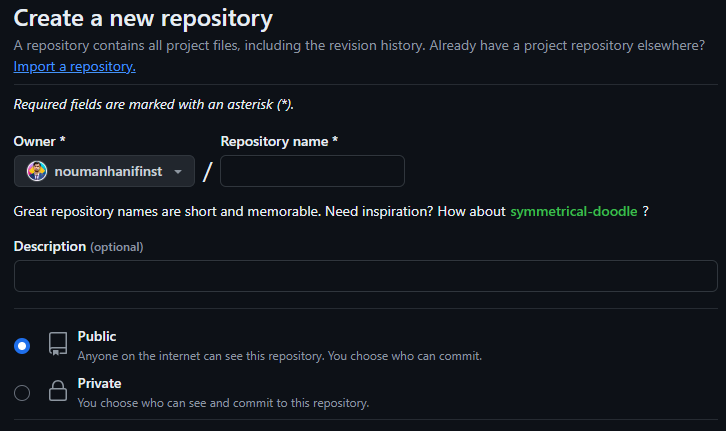


Figure 5. Creating a Repository

Which leaves us to another heading on how to create a repository.

# Create a repository

**What is a repository?**: A [*repository*](https://docs.github.com/get-started/quickstart/github-glossary#repository) is a project containing files and folders. A repository tracks versions of files and folders. For more information, see "[About repositories](https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories)" from GitHub Docs.

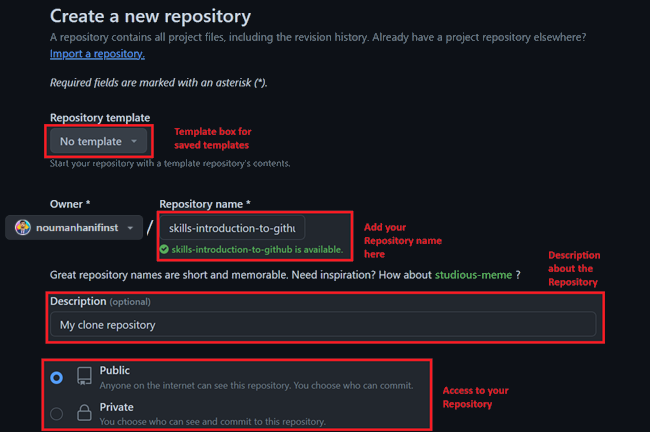
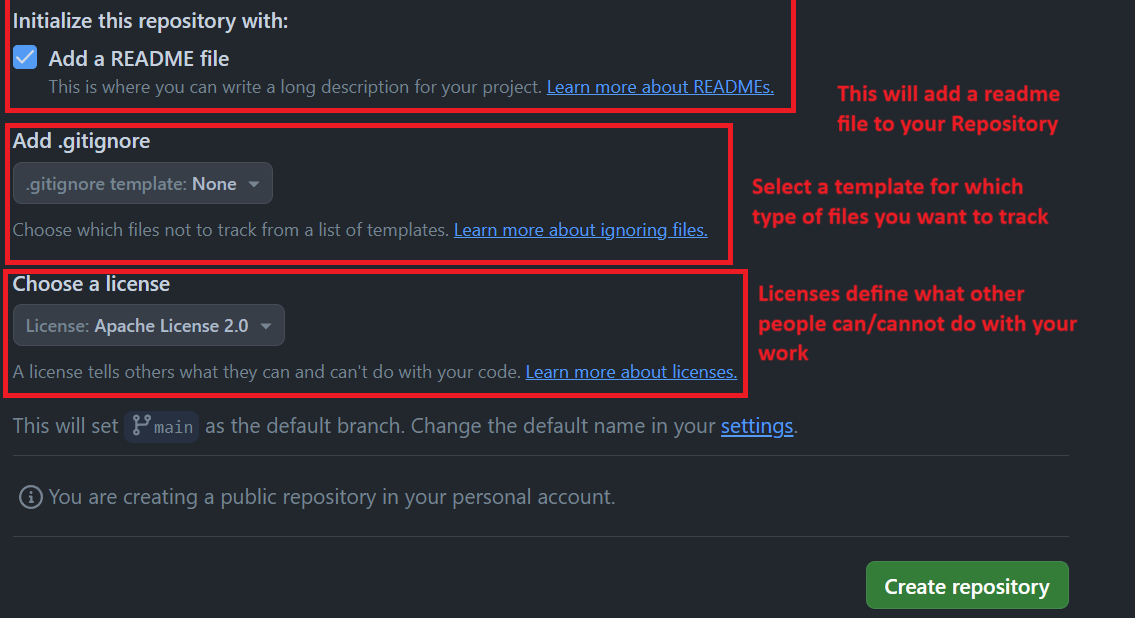


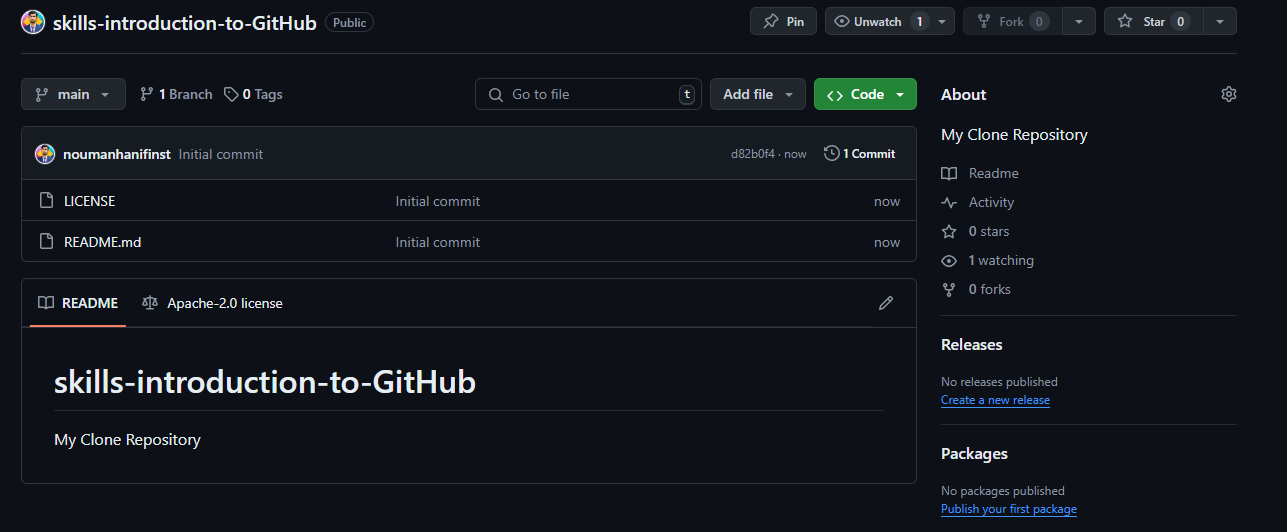
Figure 6. Explanation of creating a repository

When creating a repository you are required to fill/select at least three sections of the page. The first red box is if you want to use already saved templates (These are already made repositories with some files and branches already stored in them). The second box is your repositories name, You can name your repository anything you like but stick to proper naming conventions and name your repository according to what project you are working on, GitHub will indicate you if your repository name is unique or not. The Third box is optional and is used to display a small description about your repository. Lastly the access box and this can be crucial depending on your project. If you intend to display your project to everyone then you would need to select public access but if you choose to hide your project before releasing it to the public a private option is available and only you and those who are contributing to your project can see the repository.



**Figure 7.** Explanation of Licenses and Readme File

Lastly there also exists three more optional options when creating a repository. The First check box is a detailed description about your project, sort of like an actual MUST-READ FILE when installing pirated software. We will cover this later down in the manual on how we can modify this. The second box is for tracking which types of files to ignore, choose none and all files will be tracked otherwise you can select which type of files to exclude from tracking. Lastly a license is not compulsory for this lab but mandatory when releasing your project to the world as it defines the boundaries of what the end user can/cannot do with your work. There are many versions of this license so read and explore accordingly.  
Once all the necessary boxes are filled click on the **Create repository** button.



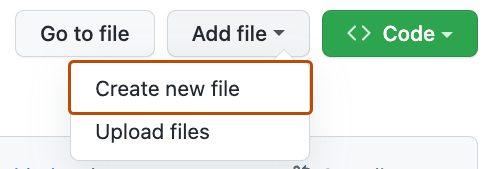
**Figure 8.** Main page of the newly created Repository

You should see a screen something like this. The bottom section is the readme file, the Top right is the description for the repository and the center is where you will see your files which you have uploaded. Right now, there are only two files as we created a license and a readme file.

# Commit a file

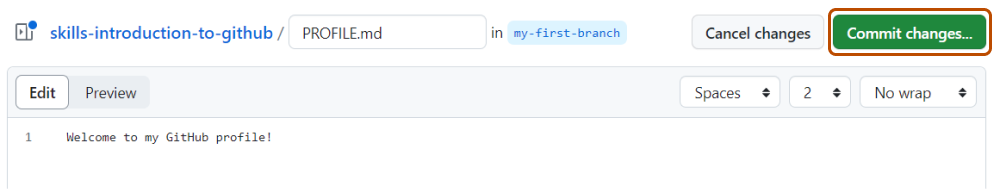
**What is a commit?** A commit is a set of changes to the files and folders in your project. A commit exists in a branch and is responsible for tracking what files were uploaded, deleted or modified in the branch.

To get started On the **< > Code** tab in the header menu of your repository, make sure you're on your new branch my-first-branch.



**Figure 9.** Option to create a new file in the Repository

Select the **Add file** drop-down and click **Create new file**.



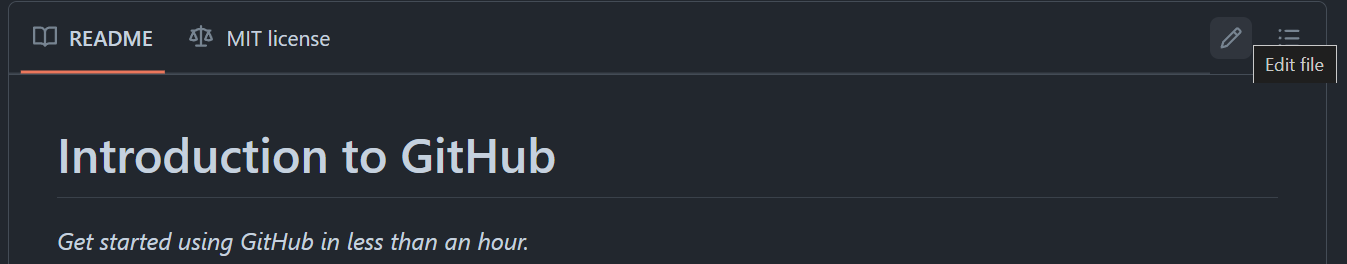
**Figure 10.** Edit preview for creating a profile.md File

In the **Name your file...** field, enter PROFILE.md.  
In the **Enter file contents here** area, write a text i.e. Programming Fundamentals Fall 2024  
Click **Commit changes...** in the upper right corner above the contents box.  
You should see a window pop something like this down below.

# Markdown language for readme Files

Markdown is a lightweight markup language that you can use to add formatting elements to plaintext text documents. GitHub uses its own flavor of Markdown, which includes some additional features beyond the standard Markdown. Here’s a guide to some of the basic elements:

Before we start click on the pencil icon in the readme section of your GitHub Repo

  
**Figure 11.** Readme file preview

Once clicked you should see a window of something like this

A screenshot of a computer

Description automatically generated

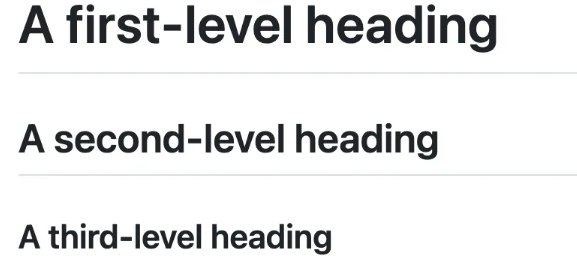
**Figure 12.** Edit preview for Repository Readme File

Here you are going to add changes so your readme file could look fancy and professional 😊. There is a preview page so whatever you change you make to your file you can see the effect in real-time.

[**Headings**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#headings)

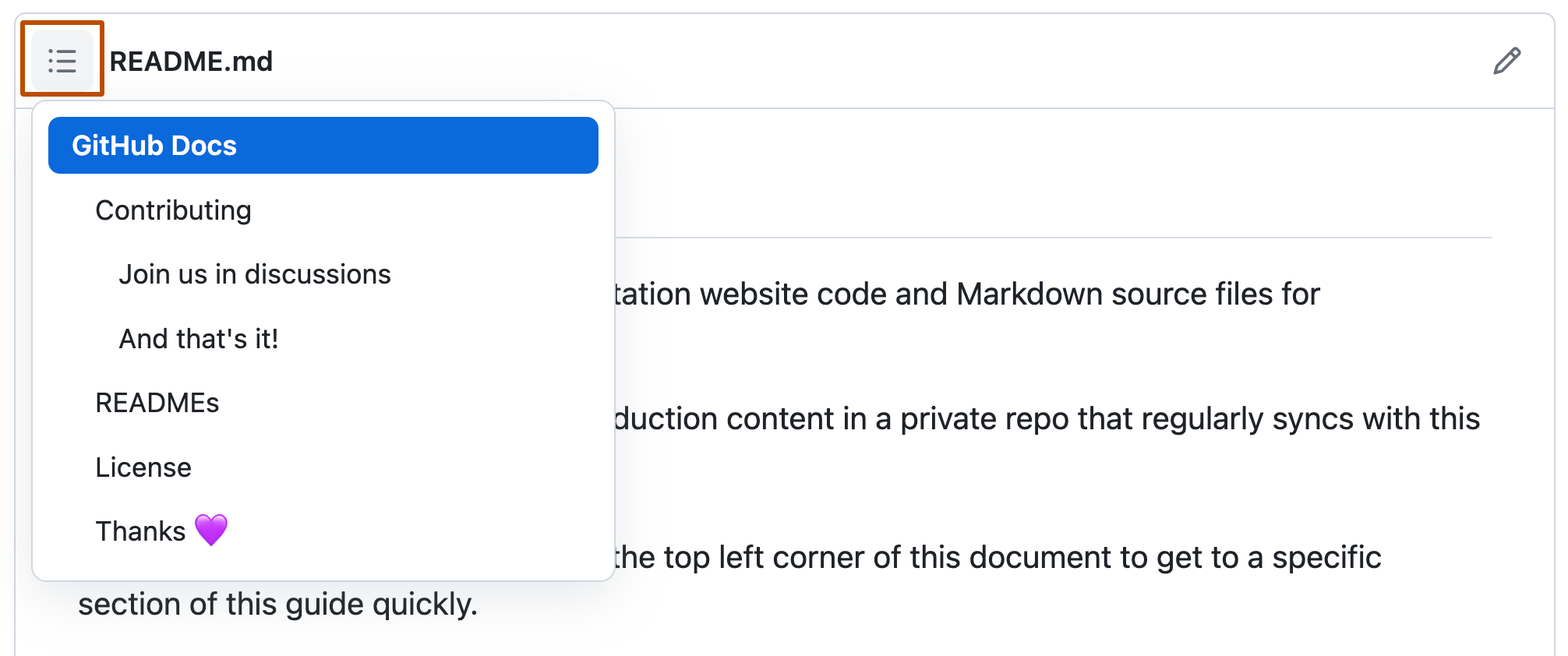
To create a heading, add one to six # symbols before your heading text. The number of # you use will determine the hierarchy level and typeface size of the heading.

1. **# A first-level heading**
2. **## A second-level heading**
3. **### A third-level heading**



**Figure 13.** Different Heading styles in Readme File

When you use two or more headings, GitHub automatically generates a table of contents that you can access by clicking   within the file header. Each heading title is listed in the table of contents, and you can click a title to navigate to the selected section.



**Figure 14.** Columns for different Headings

[**Styling text**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#styling-text)

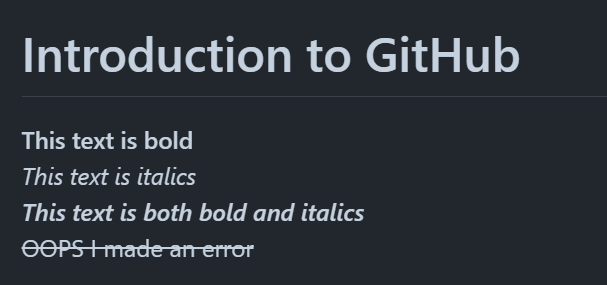
You can indicate emphasis with bold, italic, strikethrough, subscript, or superscript text in comment fields and .md files .

| **Style** | **Syntax** | **Keyboard shortcut** | **Example** | **Output** |
| --- | --- | --- | --- | --- |
| Bold | \*\* \*\* or \_\_ \_\_ | Command+B (Mac) or Ctrl+B (Windows/Linux) | \*\*This is bold text\*\* | **This is bold text** |
| Italic | \* \* or \_ \_ | Command+I (Mac) or Ctrl+I (Windows/Linux) | \_This text is italicized\_ | *This text is italicized* |
| Strikethrough | ~~ ~~ | None | ~~This was mistaken text~~ |  |
| Bold and nested italic | \*\* \*\* and \_ \_ | None | \*\*This text is \_extremely\_ important\*\* | **This text is *extremely* important** |
| All bold and italic | \*\*\* \*\*\* | None | \*\*\*All this text is important\*\*\* | ***All this text is important*** |
| Subscript | <sub> </sub> | None | This is a <sub>subscript</sub> text | This is a subscript text |
| Superscript | <sup> </sup> | None | This is a <sup>superscript</sup> text | This is a superscript text |

Example

1. # Introduction to GitHub
2. \*\*This text is bold\*\*\
3. \*This text is italics\*\
4. \*\*\*This text is both bold and italics\*\*\*\
5. ~~OOPS I made an error~~

Output

  
Note the ‘ \ ’ is used for newline. Headings do not require this.

[**Quoting text**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#quoting-text)

You can quote text with a  **>** symbol.

1. Text that is not a quote
2. > Text that is quote



[**Quoting code**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#quoting-code)

You can call out code or a command within a sentence with single backticks. The text within the backticks will not be formatted. You can also press the Command+E (Mac) or Ctrl+E (Windows/Linux) keyboard shortcut to insert the backticks for a code block within a line of Markdown.

1. Use `git status` to list all new or modified files that haven't yet been committed.
2. Some basic Git commands are:
3. ```
4. git status
5. git add
6. git commit
7. **```**

A screen shot of a computer

Description automatically generated

**Figure 15.** Example of Newline items.

[**Lists**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#lists)

You can make an unordered list by preceding one or more lines of text with -, \*, or +.

1. - George Washington
2. \* John Adams
3. + Thomas Jefferson



To order your list, precede each line with a number.

1. 1. James Madison
2. 2. James Monroe
3. 3. John Quincy Adams



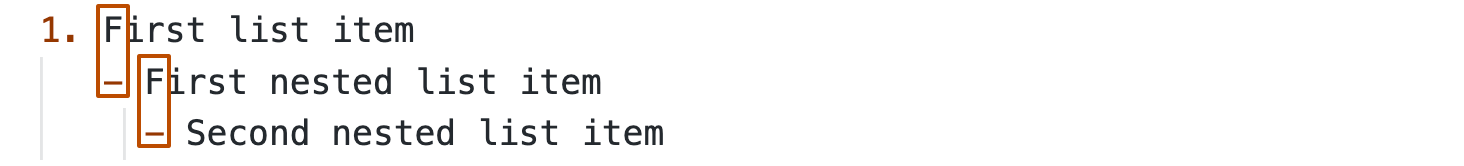
[**Nested Lists**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#nested-lists)

You can create a nested list by indenting one or more list items below another item.

To create a nested list using the web editor on GitHub or a text editor that uses a monospaced font, like [Visual Studio Code](https://code.visualstudio.com/), you can align your list visually. Type space characters in front of your nested list item until the list marker character (- or \*) lies directly below the first character of the text in the item above it.

1. 1. First list item
2. - First nested list item
3. - Second nested list item

Output

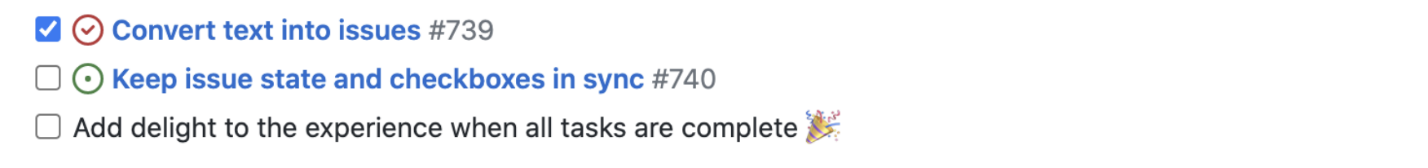




[**Task lists**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#task-lists)

To create a task list, preface list items with a hyphen and space followed by [ ]. To mark a task as complete, use [x].

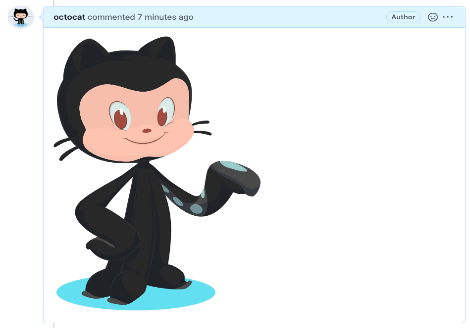
1. - [x] #739
2. - [ ] https://github.com/octo-org/octo-repo/issues/740
3. - [ ] Add delight to the experience when all tasks are complete :tada:



[**Images**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#images)

You can display an image by adding ! and wrapping the alt text in [ ]. Alt text is a short text equivalent of the information in the image. Then, wrap the link for the image in parentheses ().

1. ![Screenshot of a comment on a GitHub issue showing an image, added in the Markdown, of an Octocat smiling and raising a tentacle.](<https://myoctocat.com/assets/images/base-octocat.svg>)



**Figure 16.** Displaying a Picture in Readme File

Note that a direct weblink and link to your repository path where the image is located can be utilized in this manner.

Some Problems to Solve

* Create your repository with your roll number being your repo name
* Upload the algorithms and pseudo codes in your repository
* Create a small intro about yourself in the readme file with pictures and bullet points