



CL1002
*Programming
Fundamentals Lab*

Lab 02
Introduction to
Pseudocode,
Algorithms and
GitHub

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Add Semester Info here.

AIMS AND OBJECTIVES

The purpose of this lab is to understand what pseudo codes and algorithms of a program are, what are its specific keywords and its operations and a brief introduction to what GitHub is.

INTRODUCTION

In this lab we will introduce the following concepts.

Flowcharts (Continued from previous Lab)

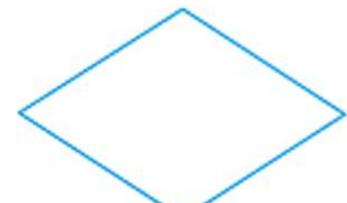
Pseudocode

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.



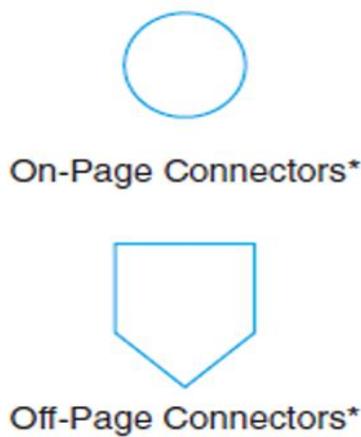
Decision



Process Module

The diamond indicates a decision. It has one entrance and two and only two exits from the block. One exit is the action when the resultant is *True* and the other exit is the action when the resultant is *False*.

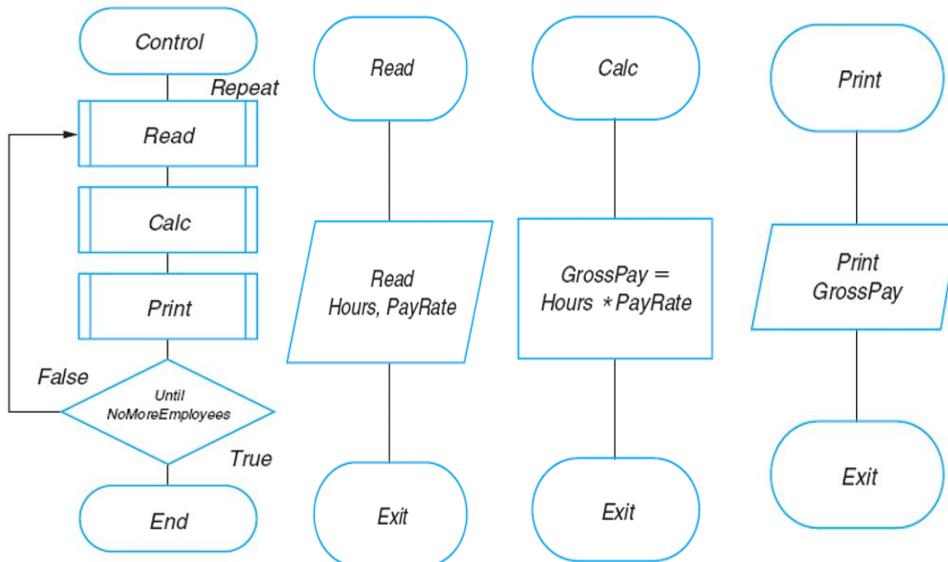
Rectangles with lines down each side indicate the process of modules. They have one entrance and only one exit.



Flowchart sections can be connected with two different symbols. The circle connects sections on the same page, and the home base plate connects flowcharts from page to page. Inside these two symbols the programmer writes letters or numbers. The on-page connector uses letters inside the circle to indicate where the adjoining connector is located. An *A* connects to an *A*, a *B* to a *B*, etc. The off-page connectors use the page number where the next part or the previous part of the flowchart is located. This allows the reader to easily follow the flowchart. On- and off-page connectors will have either an entrance or an exit.

EXAMPLE

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



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 in a logistics company responsible for delivering packages. Design a flowchart to manage the process of receiving, sorting, and delivering packages. Include decision structures for handling fragile items and urgent deliveries.
2. Imagine you are automating the process of a vending machine. Create a flowchart that includes decision points for user input, selecting products, accepting payment, and dispensing the correct item. Include error-handling for invalid inputs and insufficient funds.

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

```
01      START
02
03      // Input/Output
04      INPUT number1
05      INPUT number2
06
07      // variables and Initialization
08      SET sum to 0
09
10      // Process Steps
11      SET sum to number1 + number2
12
13      // Conditional Statements
14      IF sum > 0 THEN
15          PRINT "The sum is positive"
16      ELSE
17          PRINT "The sum is non-positive"
18      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.

1. Write pseudocode to find the smallest number among three given variables. Implement a decision-making structure to compare the variables.
2. Create pseudocode to subtract two numbers without using the - operator. (Hint: Use addition and complement techniques.)
3. Develop pseudocode for a basic calculator that performs multiplication and division. The pseudocode should prompt the user for two numbers and an operator, then display the result of the operation.

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:

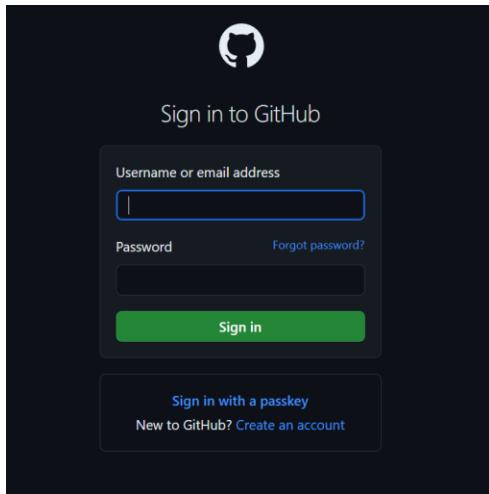
1. Write an algorithm to determine whether a number is a prime number. The algorithm should iterate through possible divisors and determine if the number has any divisors other than 1 and itself.
2. Create an algorithm that asks the user for a day number (1-365) and outputs the corresponding day of the week, assuming that January 1st is a Monday.
3. Develop an algorithm for a program that takes two numbers as input and finds the Greatest Common Divisor (GCD) of the two numbers using the Euclidean algorithm.

GITHUB

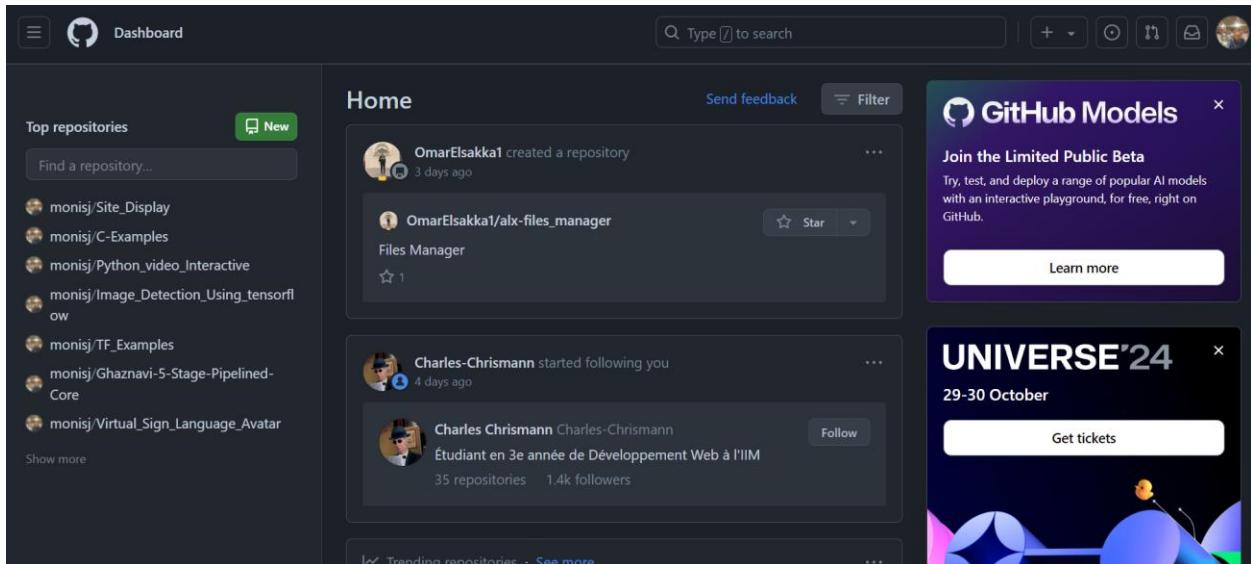
GitHub is a web-based platform that provides version control using Git, allowing multiple people to collaborate on projects, track changes, and manage code repositories efficiently. It's widely used by developers for both open-source and private projects.

CREATE AN ACCOUNT ON GITHUB

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



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.



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.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner * Repository name *

 monisj /

Great repository names are short and memorable. Need inspiration? How about [stunning-octo-lamp](#) ?

Description (optional)

 **Public**
Anyone on the internet can see this repository. You choose who can commit.

 **Private**
You choose who can see and commit to this repository.

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

CREATE A REPOSITORY

What is a repository?: A [repository](#) is a project containing files and folders. A repository tracks versions of files and folders. For more information, see "[About repositories](#)" from GitHub Docs.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Repository template

No template ▾

Template box for saved templates

Start your repository with a template repository's contents.

Owner * monisj / Repository name *

skills-introduction-to-github

✓ skills-introduction-to-github is available.

Add your Repository name here

Description about the Repository

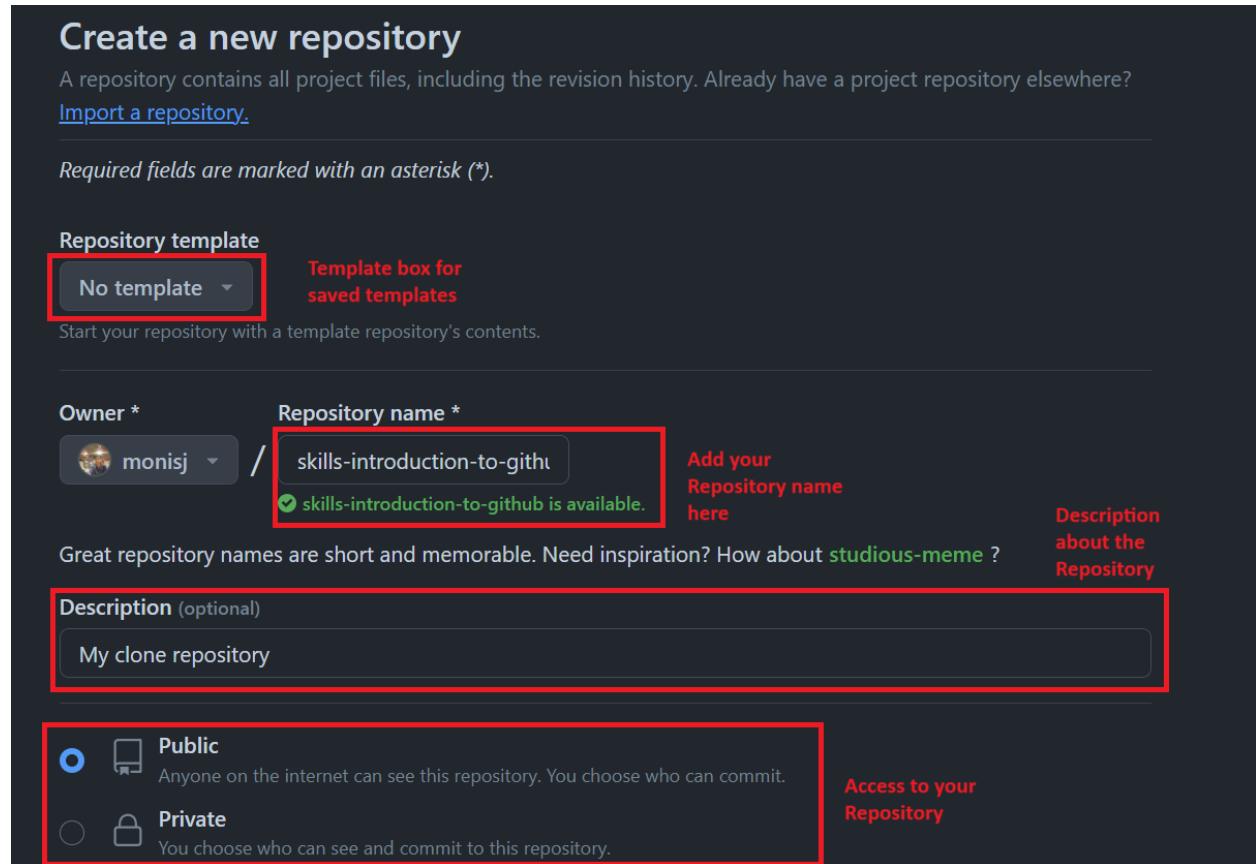
Description (optional)

My clone repository

Public Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Access to your 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.

Initialize this repository with:

Add a README file
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
.gitignore template: **None** ▾
Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

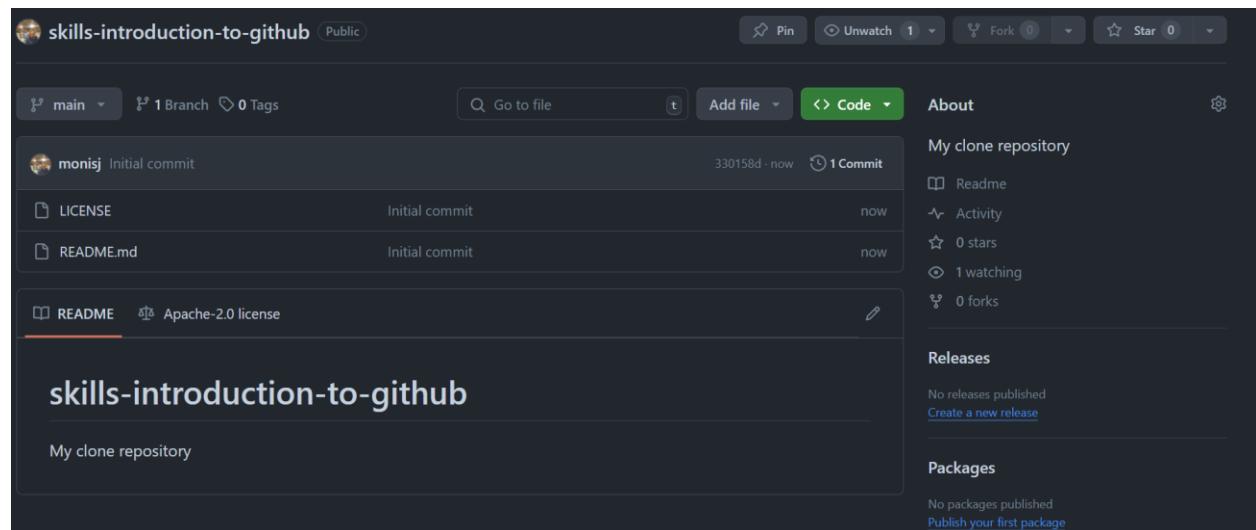
Choose a license
License: **Apache License 2.0** ▾
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

ⓘ You are creating a public repository in your personal account.

Create repository

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.

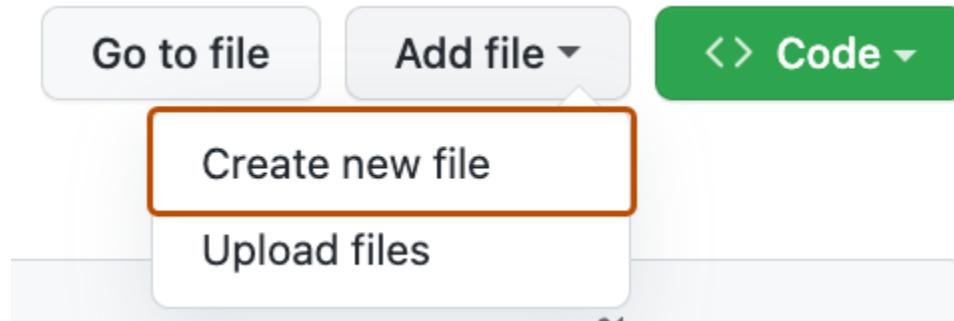


You should see a screen something like this. The bottom section is the readme file, the Top right is the description for the repository and the center is where you will see your files which you have uploaded. Right now, there are only two files as we created a license and a readme file.

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.



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

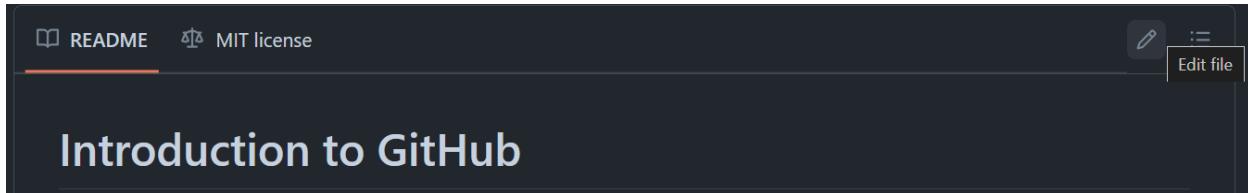
Click **Commit changes...** in the upper right corner above the contents box.

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



Once clicked you should see a window of something like this

A screenshot of a GitHub code editor interface. The top bar includes 'Edit', 'Preview', and a GitHub Copilot icon. On the right, there are settings for 'Spaces', '2', and 'Soft wrap'. The main editor area shows the line '1 # Introduction to GitHub'. A status bar at the bottom indicates keyboard shortcuts for navigating the page.

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

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.

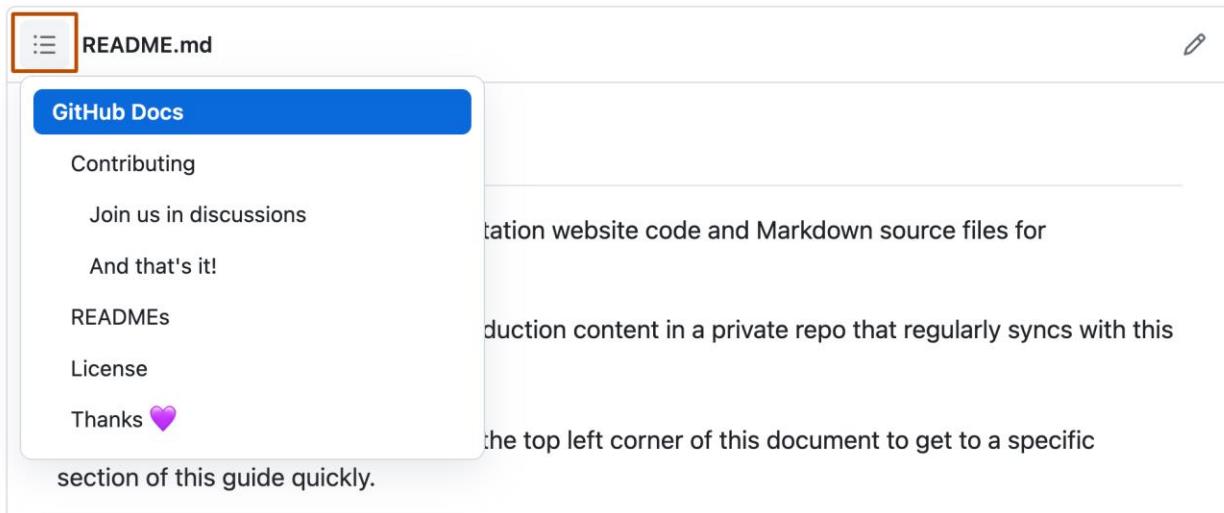
```
19     # A first-level heading
20     ## A second-level heading
21     ### A third-level heading
```

A first-level heading

A second-level heading

A third-level heading

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.



The screenshot shows a GitHub README.md file. At the top left is a navigation bar with icons for Home, GitHub Docs (which is highlighted with a blue bar), Search, and Issues. Below the navigation bar is the file name "README.md". The main content area contains several sections: "Contributing", "Join us in discussions", "And that's it!", "READMEs", "License", and "Thanks ❤️". To the right of these sections, there is a preview of the file's content. At the bottom left, there is a note: "Click the top left corner of this document to get to a specific section of this guide quickly."

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** <u>This text is italicized</u>	This is bold text <i>This text is italicized</i>
Italic	* * or _	Command+I (Mac) or Ctrl+I (Windows/Linux)	_This text is italicized_	<i>This text is italicized</i>
Strikethrough	~~ ~~	None	~~This was mistaken text~~	This was mistaken text
Bold and nested italic	** ** and __	None	**This text is <u>extremely</u> important**	This text <i>extremely important</i>
All bold and italic	*** ***	None	***All this text is important***	<i>All this text is important</i>
Subscript	_{ }	None	This is a _{subscript} text	This is a _{subscript} text
Superscript	^{ }	None	This is a ^{superscript} text	This is a ^{superscript} text

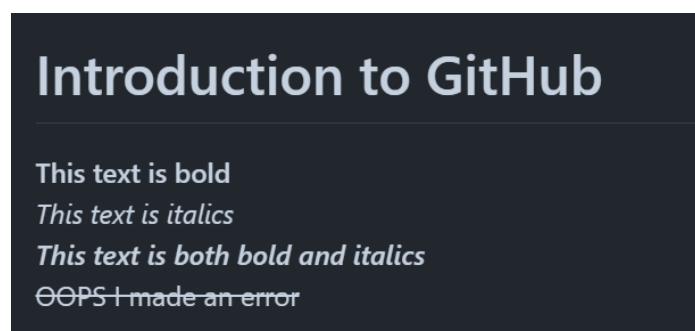
Example

```

01  # Introduction to GitHub
02  **This text is bold**\
03  *This text is italics*\
04  ***This text is both bold and italics***\
05  ~~OOPS I made an error~~

```

Output



Note the '\ ' is used for newline. Heading do not require this.

Quoting text

You can quote text with a > symbol.

```
01 Text that is not a quote  
02 > Text that is quote
```

Text that is not a quote

Text that is a quote

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.

```
01 Use `git status` to list all new or modified files that haven't yet been committed.  
02 Some basic Git commands are:  
03  
04 git status  
05 git add  
06 git commit  
07
```

Use `git status` to list all new or modified files that haven't yet been committed. Some basic Git commands are:

```
git status  
git add  
git commit
```



Lists

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

```
01 - George Washington
02 * John Adams
03 + Thomas Jefferson
```

- George Washington
- John Adams
- Thomas Jefferson

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

```
01 1. James Madison
02 2. James Monroe
03 3. John Quincy Adams
```

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

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](#), 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.

```
01 1. First list item
02     - First nested list item
03         - Second nested list item
```

Output

```
1. First list item
  - First nested list item
    - Second nested list item
```

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

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].

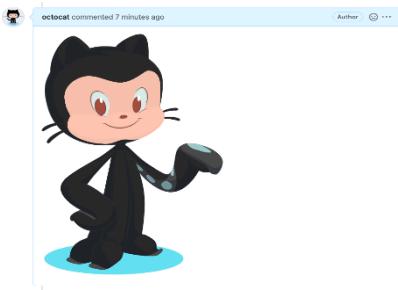
```
04      - [x] #739
05      - [ ] https://github.com/octo-org/octo-repo/issues/740
06      - [ ] Add delight to the experience when all tasks are
complete :tada:
```

- Convert text into issues #739
- Keep issue state and checkboxes in sync #740
- Add delight to the experience when all tasks are complete 

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 ()).

```
01 ! [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)
```



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

Some Problems to Solve

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