Tab 1

Hello AI: Introduction to AI & Setting Up Dev Tool

Code: AIEPCM1L1

Outline :

Start your AI adventure with a Python refresher! Explore AI's thrilling history and real-world impact. In this first lesson, you'll dive back into programming with Python, set up essential tools like Visual Studio Code, Git, and GitHub, and begin building the foundation for your own AI projects.

Overview :

Topics introduced

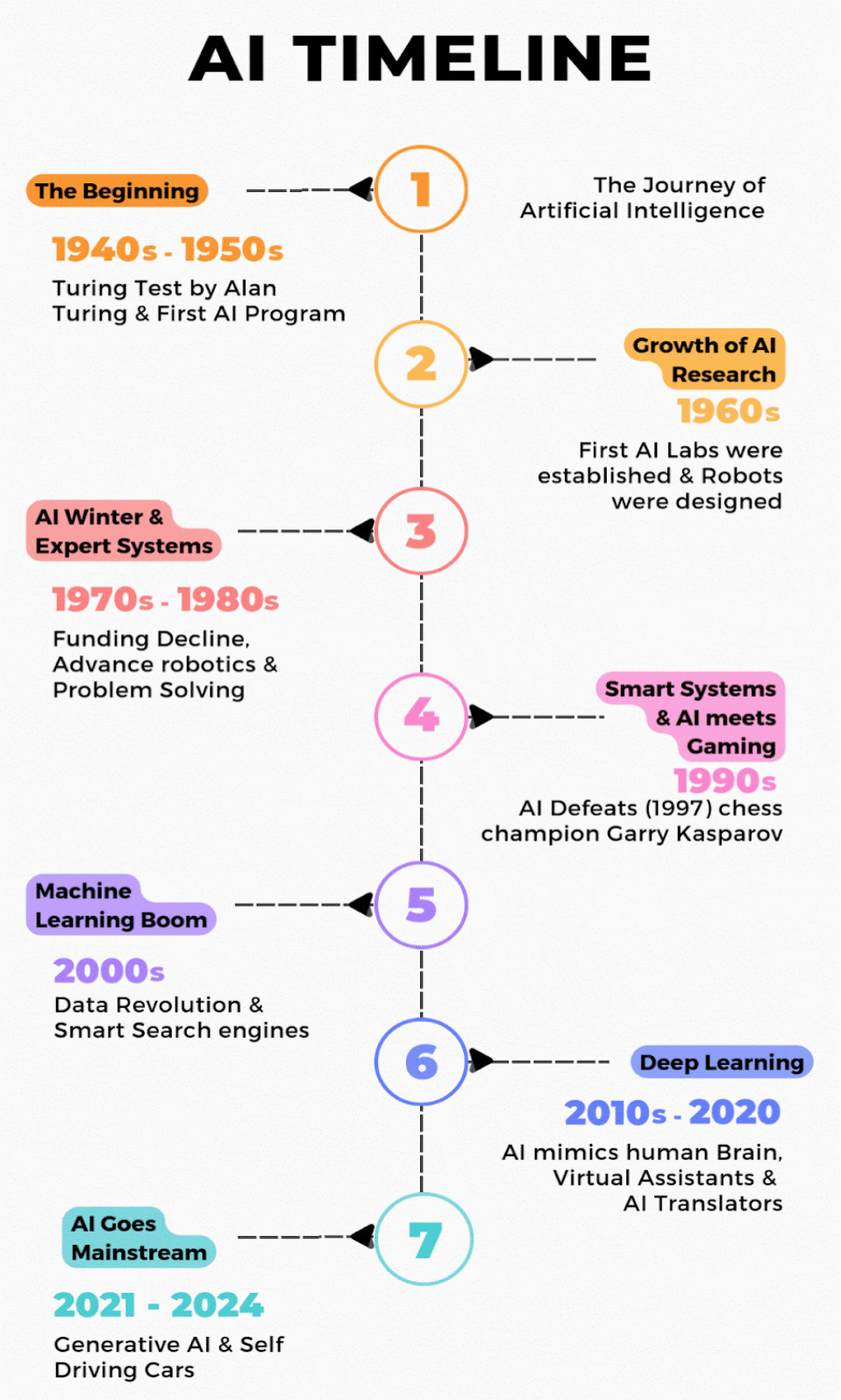
- Introduction to AI
  - History of AI
- What is programming?
  - Programming language
  - Introduction to Python
  - Applications of Python
  - Setting up Python
- Setting up Visual Studio Code
- Git & GitHub
- Folder Structure
- Install & Use extensions in VS Code
- Git config
- Push code to GitHub
- Algorithm
- Flowchart
- Python refresher

Topics in detail

Introduction to AI

Artificial Intelligence (AI) is like giving computers and machines a form of intelligence that helps them think and learn similarly to humans. Imagine teaching a computer to understand images, recognize speech, make smart decisions, and even learn from its mistakes. Just as you learn from experience - like getting better at a sport with practice - AI systems improve by processing large amounts of data and finding patterns. They power everyday tools like Siri, recommending your next Netflix show, help cars drive themselves, and even assist doctors in diagnosing diseases. While regular computers just follow strict instructions, AI can adapt and figure things out on its own.

History of AI



The beginning (1940s - 1950s)

- 1943: First artificial neurons proposed by McCulloch and Pitts
- Turing Test: Alan Turing proposed a test to see if machines could "think" like humans. This test became the first step toward AI.
- 1956: Term "Artificial Intelligence" coined at Dartmouth Conference

Growth of AI Research (1960s)

- First AI Labs: Universities like MIT and Stanford created AI research labs, studying how machines could mimic human language and reasoning.First chatbots and problem-solving programs created
- Robots: Simple robots were designed to follow instructions and navigate spaces.Robots begin to appear in factories
- Early pattern recognition systems developed

AI Winter & Expert Systems (1970s - 1980s)

- Funding Decline: Many thought AI was over-promised and under-delivered, so funding was cut, leading to slow progress.
- Renewed Interest: AI found new purpose with "expert systems," which could make decisions in specialized fields, like diagnosing diseases.
- Robotics Advances: Robots could now perform tasks like assembly in factories.

AI meets Gaming & Smart Systems (1990s)

- Chess Match: IBM’s Deep Blue defeated world chess champion Garry Kasparov in 1997, showing that AI could tackle complex tasks.
- Better speech recognition in computers
- Robot pets and basic home robots appear
- Internet search engines become smarter

Machine Learning Boom (2000s)

- Data Revolution: With more data and computing power, machine learning allowed AI to learn from patterns and improve performance.
- Real-World Applications: AI started powering search engines, recommendation systems, and more.
- IBM Watson wins Jeopardy! (2011)
- Siri, Google Now, and other digital assistants emerge

- Self-driving cars begin development
- Better image recognition technology

Deep learning

- Deep Learning: A branch of machine learning that mimics the human brain. AI could now recognize images, translate languages, and even understand voice commands.
- Everyday AI: Tools like Siri, Alexa, and Google Translate brought AI to millions.
- AlphaGo beats world champion at Go (2016)
- Virtual assistants become more common
- AI helps in medical diagnosis and drug discovery
- Better language translation tools

AI Goes Mainstream

- Generative AI: New models like ChatGPT can generate text, images, and even music.
- Self-Driving Cars & Robotics: AI is now part of cars, healthcare, and industries worldwide.
- ChatGPT and large language models emerge
- AI art generators like DALL-E appear
- AI helps in scientific research
- Debates about AI safety and ethics become important
- AI tools become widely available for everyday use

What programming?

Programming is the process of creating instructions that tell a computer how to perform tasks. These instructions, written in code, guide the computer in doing everything from simple calculations to complex operations.

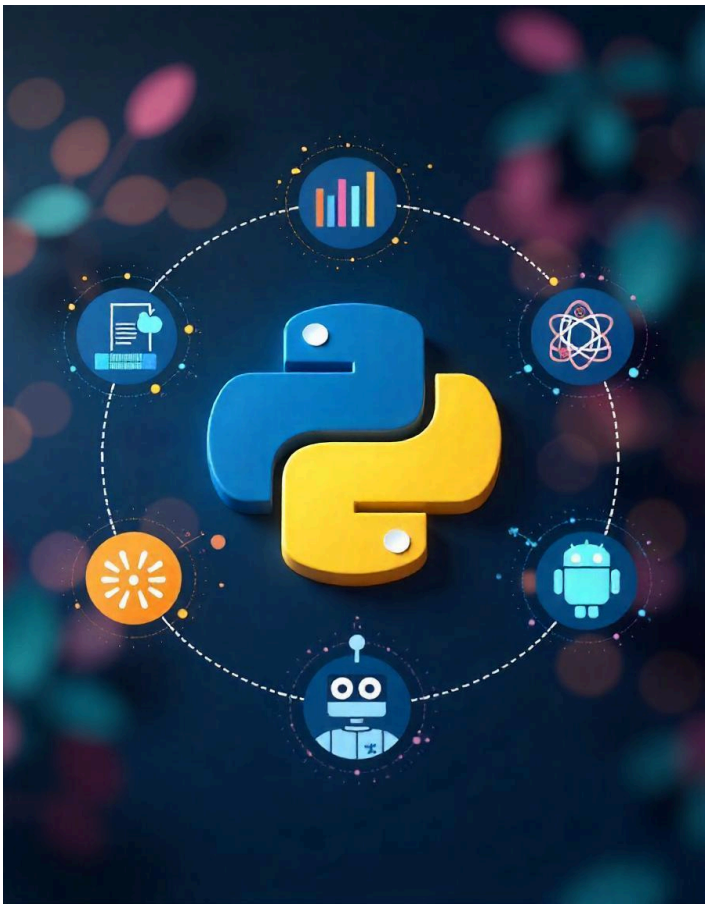
Programming language

A programming language is a language used by programmers to write code. This code is then understood and executed by computers to carry out specific tasks and solve problems.

Introduction to Python

Python is an interpreted, object-oriented, high-level programming language. Python is simple to write and easy to learn. The syntax emphasises readability and therefore reduces the cost of program maintenance.

Applications of Python



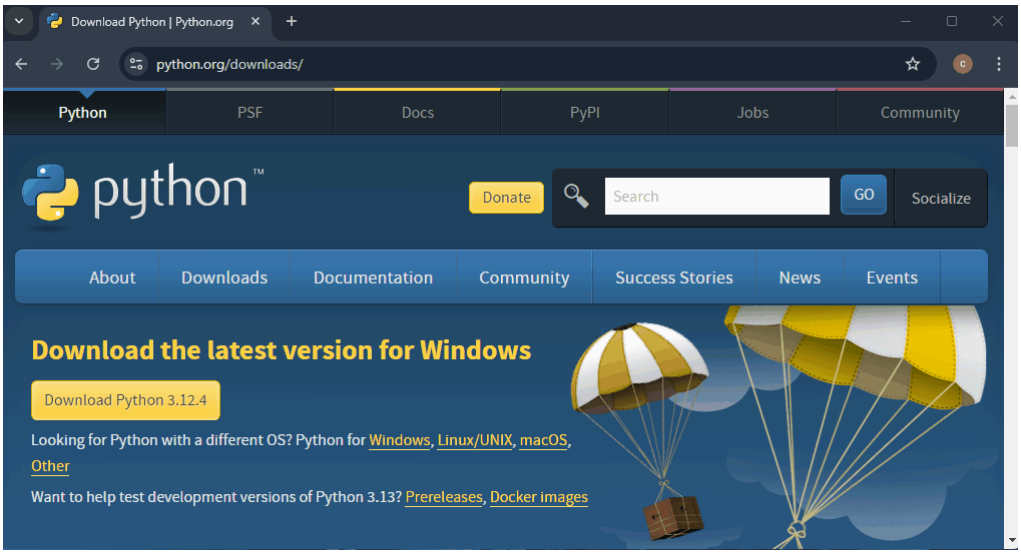
Web Development: Building websites and web applications using Django/Flask frameworks
Data Science: Analyzing and visualizing complex data using pandas/numpy
AI/Machine Learning:
- Deep Learning: Neural networks for complex pattern recognition using PyTorch/TensorFlow
- Natural Language Processing: Text analysis, chatbots, and language translation
- Computer Vision: Image/video processing, object detection
- Generative AI: Creating text, images, and code using models like GPT
- Reinforcement Learning: Training AI agents through reward-based learning

Automation: Scripting repetitive tasks and system processes
Game Development: Creating 2D/3D games with PyGame

IoT: Programming smart devices and microcontrollers
Cybersecurity: Network security tools and ethical hacking
Scientific Computing: Complex mathematical simulations
Desktop Applications: GUI apps using tkinter/PyQt
API Development: Creating web services and integrations

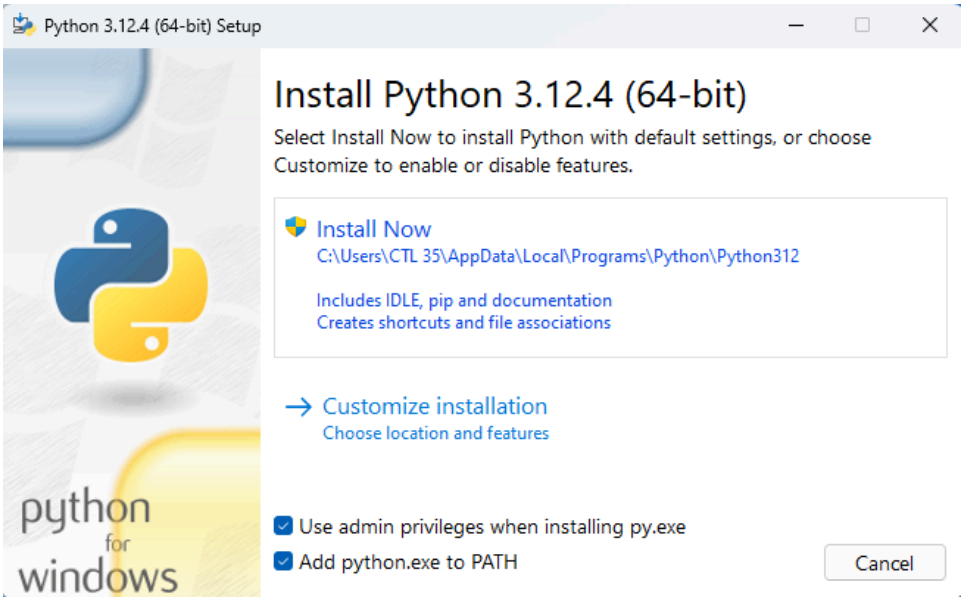
Download Python

Click here : Link, Once the download is complete open the file

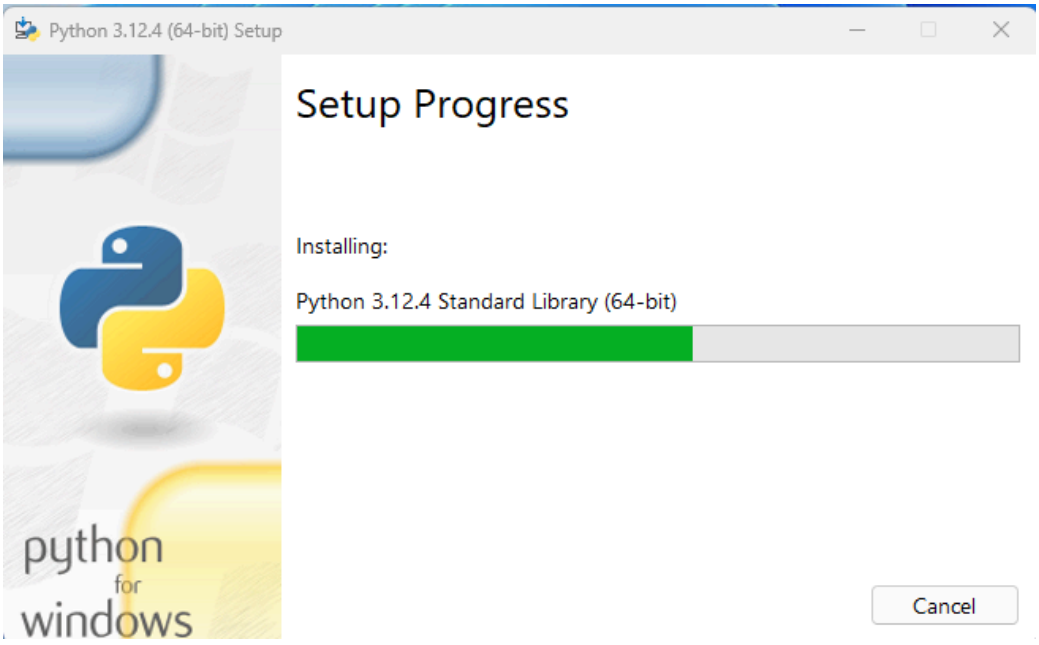


Install Python.

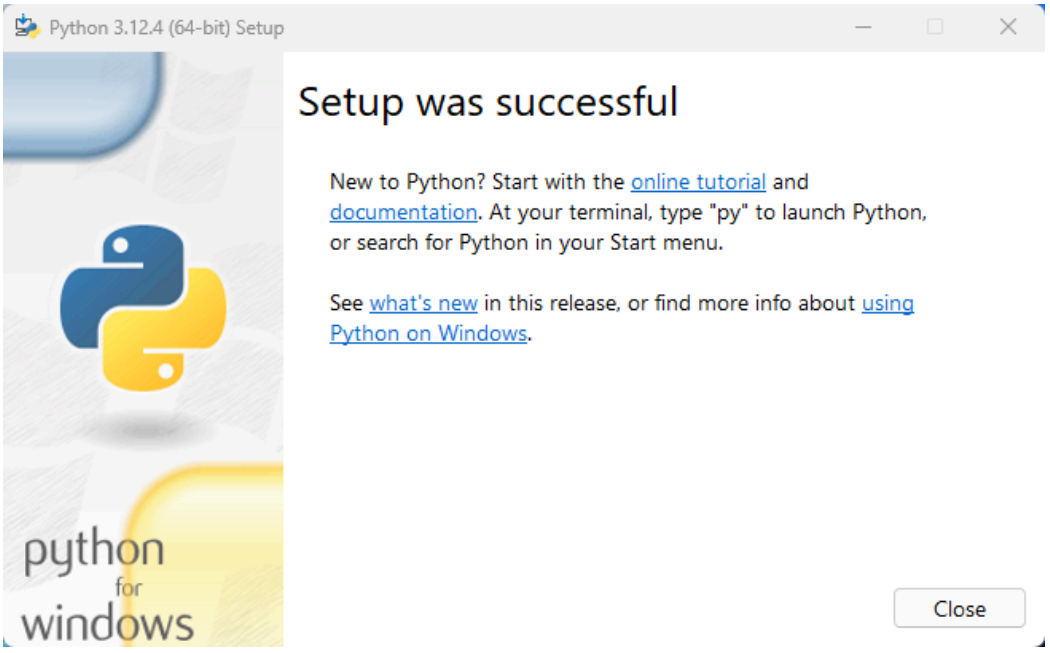
Step 1 : make sure the check boxes are marked as shown in the screenshot & Click on Install Now



Step 2: Wait for the installation to complete



Step 3: Click on Close



Setting up VS Code

What is VS Code

Visual Studio Code (VS Code) is a free code editor created by Microsoft. It helps you write and edit code easily with many features like:

- Text Editing: Write code with helpful tools like auto-complete and error checking.
- Extensions: Add features to make coding easier, like live previews of your web pages and sharing your code with others in real time.
- Debugging: Find and fix errors in your code.
- Customizable: Change the look and feel to match your preferences.

VS Code supports many programming languages and is a great tool for learning and developing websites and apps.

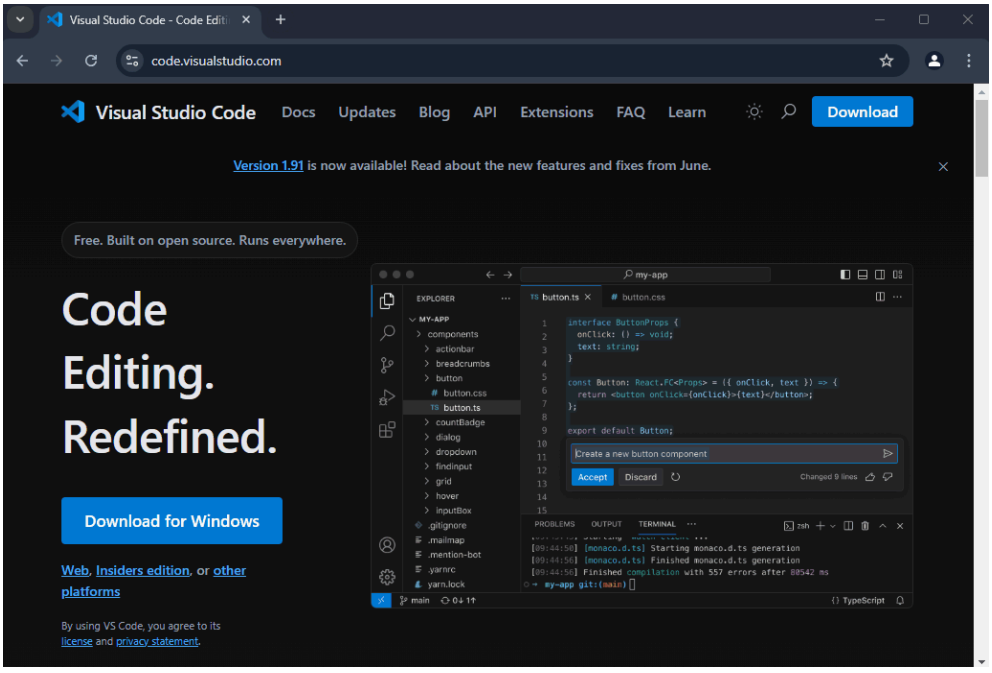
Download VS Code

Download VS Code as shown in the following GIF

Website Link: [Click here]

Chromebook Reference: [Click here]

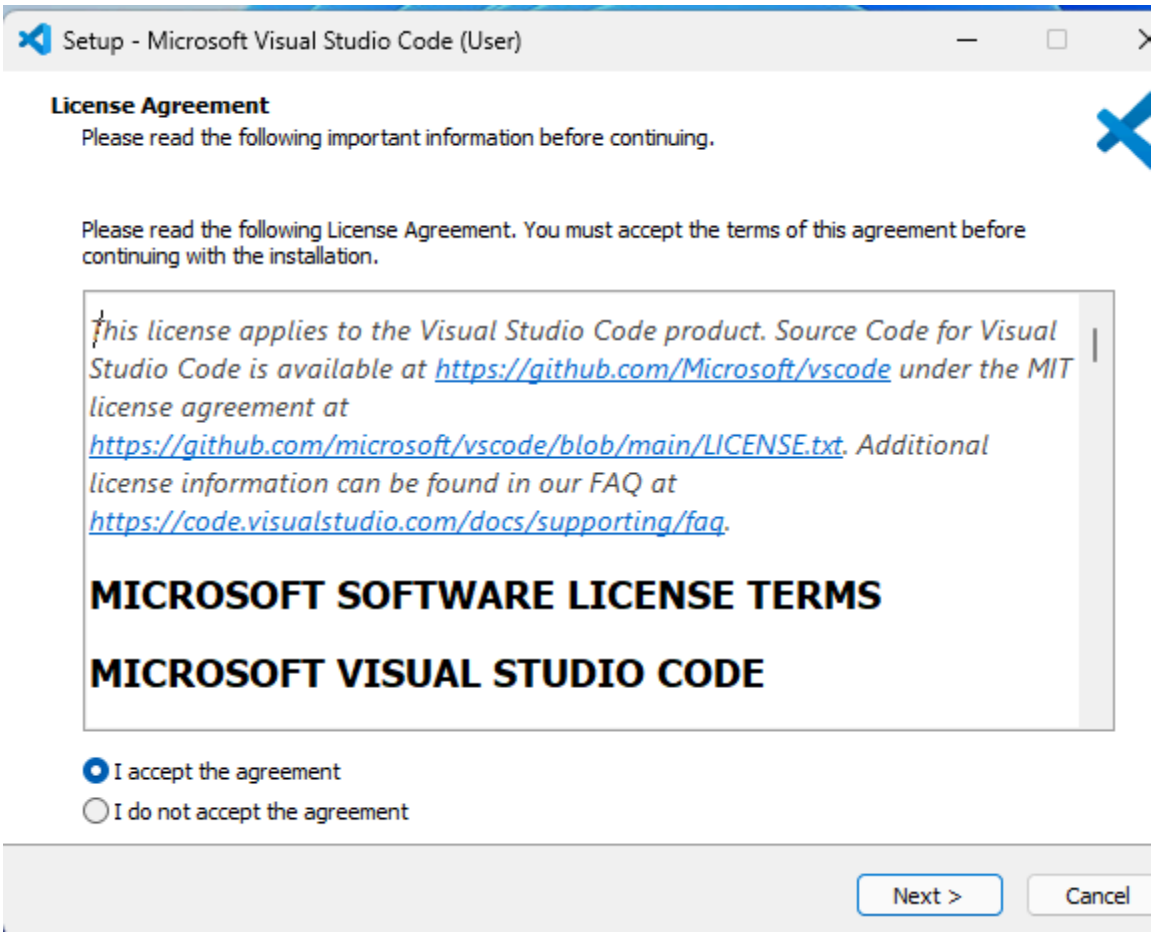
Mac Reference: [Click here]



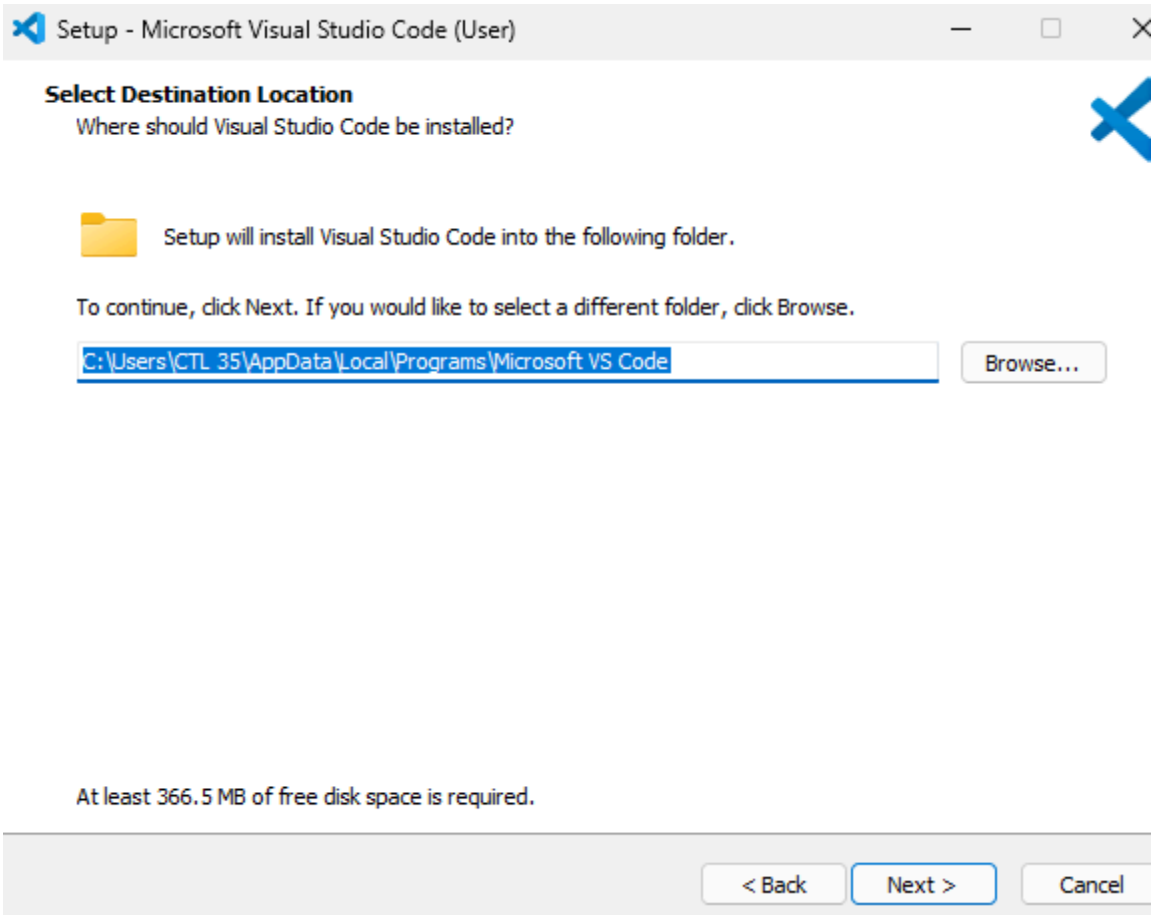
Install VS Code

Allow the software to run for installation & follow each step as shown

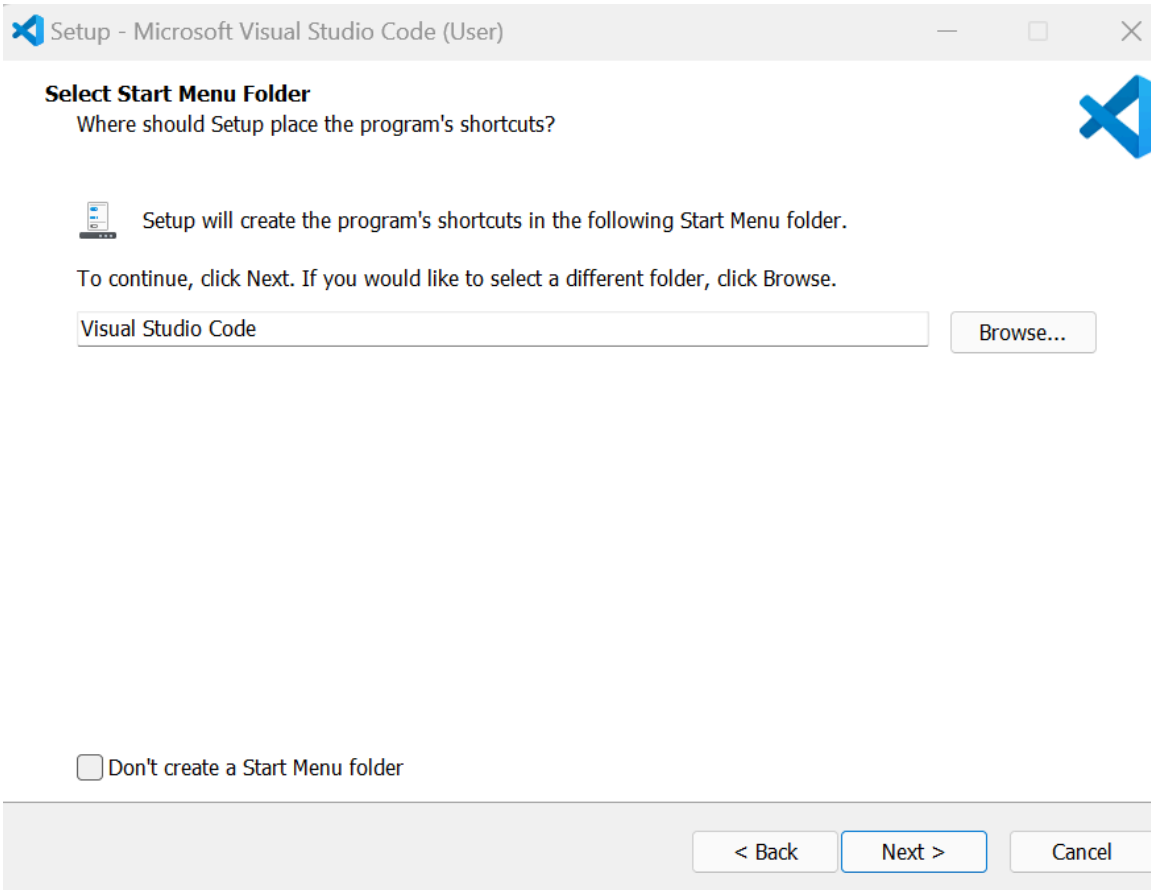
Step 1 : Select I accept the agreement & Click on Next



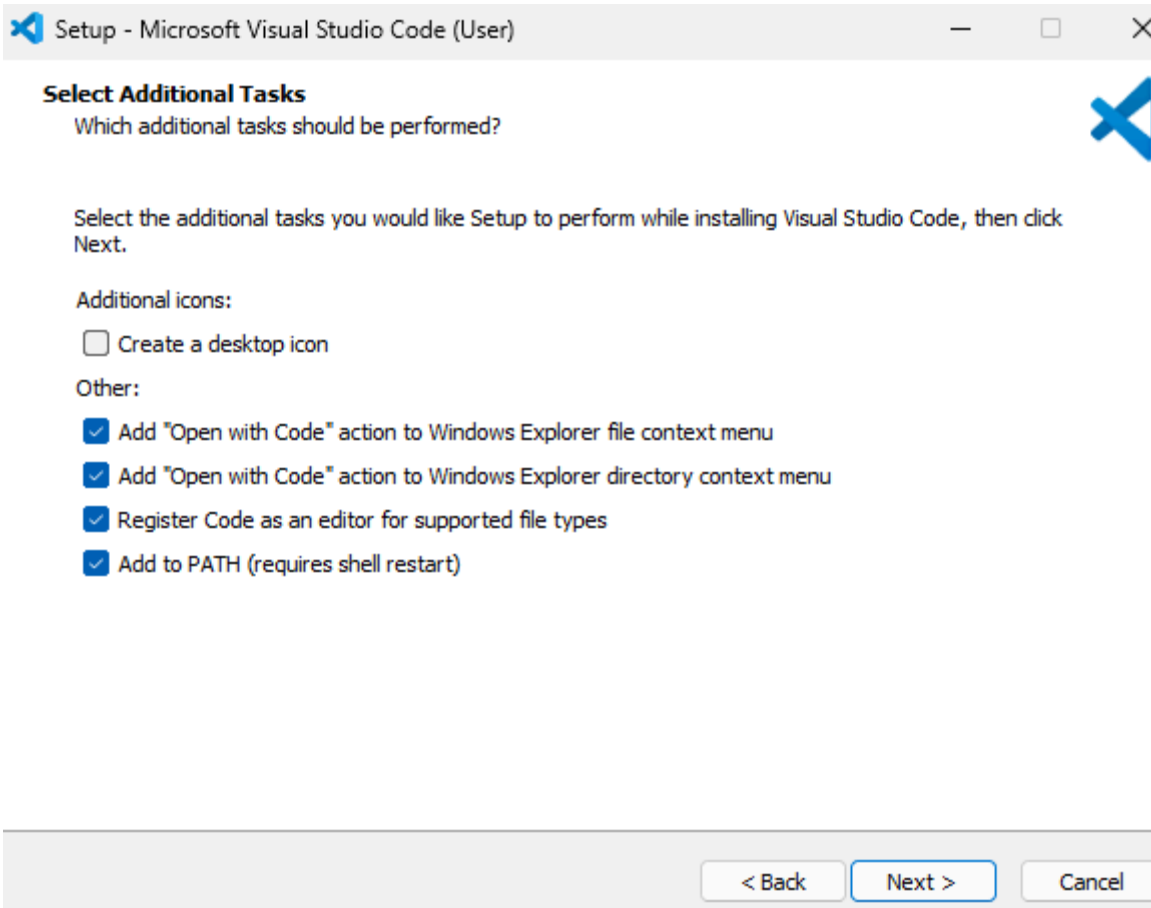
Step 2 : Click on Next



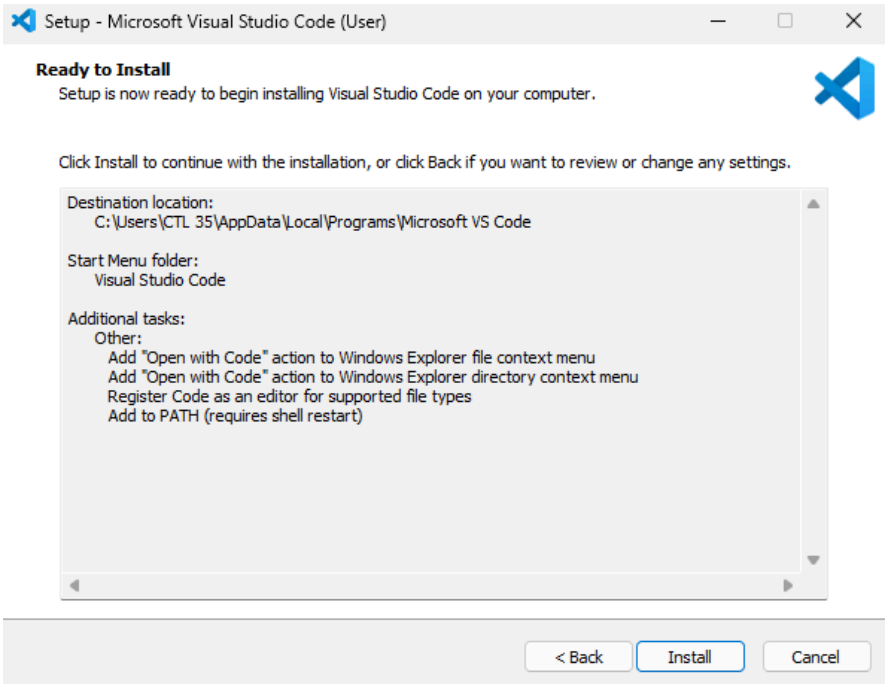
Step 3: Click on Next



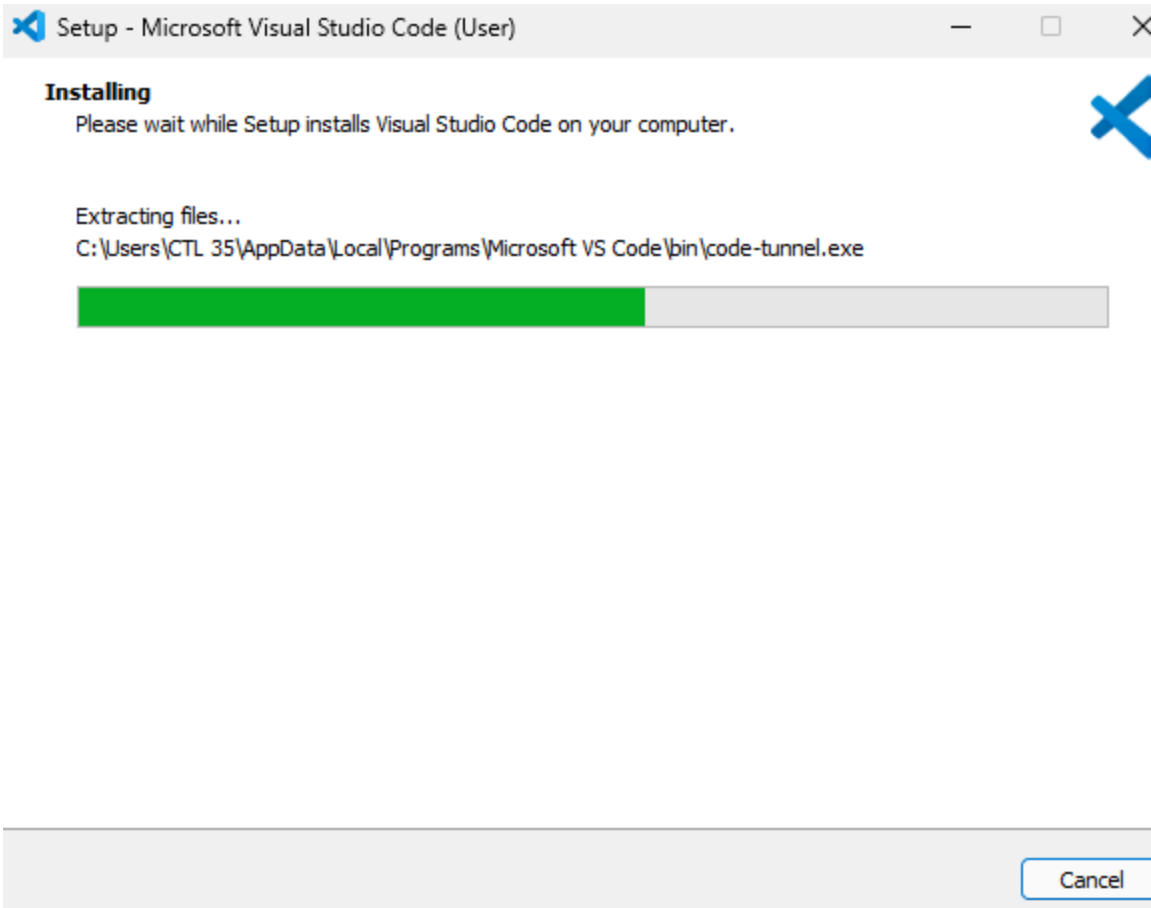
Step 4 : Match the checkboxes as shown in the



Step 5: Click On Install

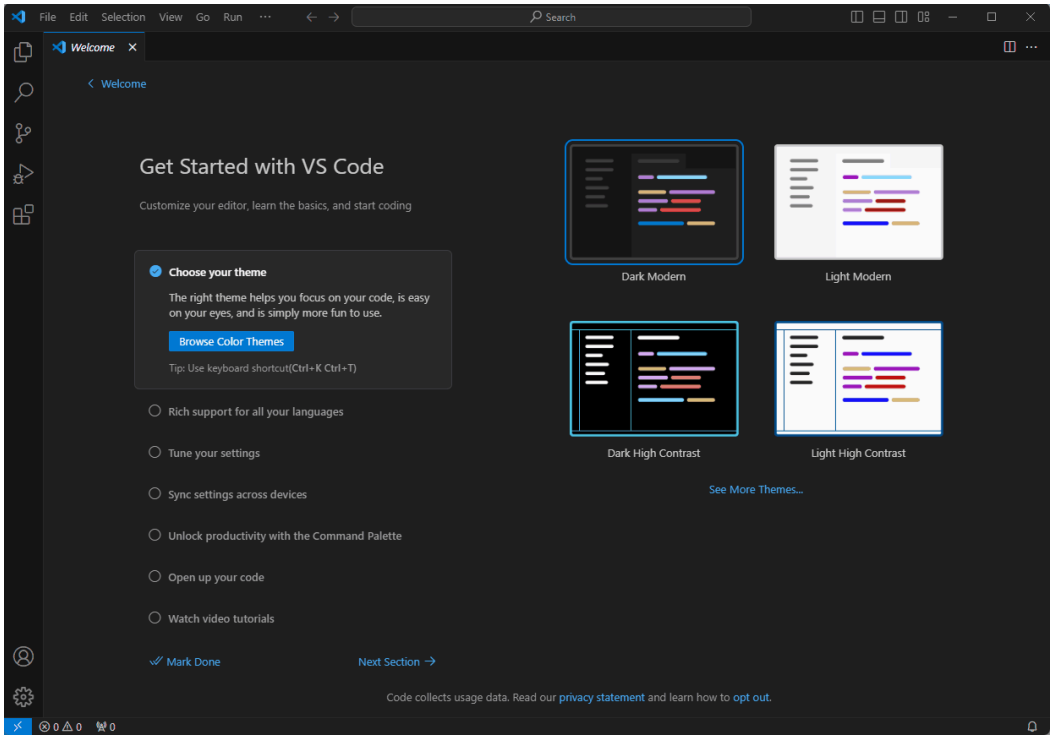


Step 6 : wait for the installation to complete and click on Finish

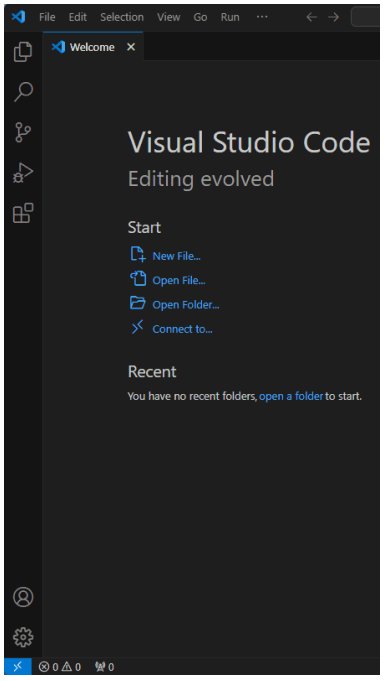


Basic Visual Studio Code settings

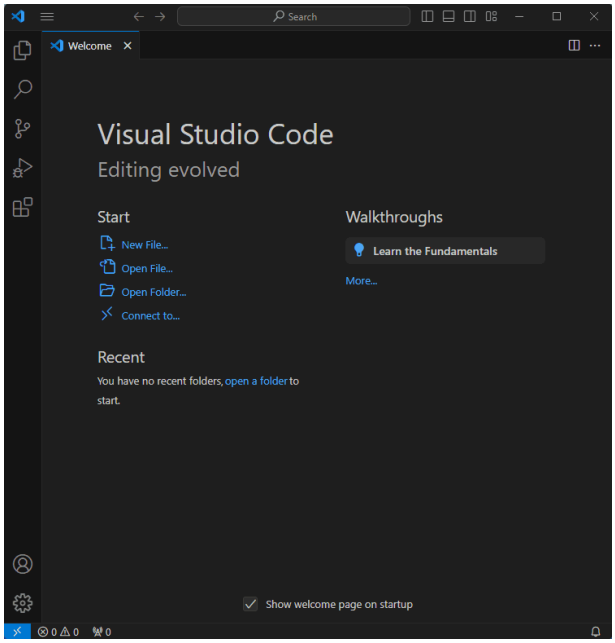
Open VS Code Select desired Theme => Click on Mark Done



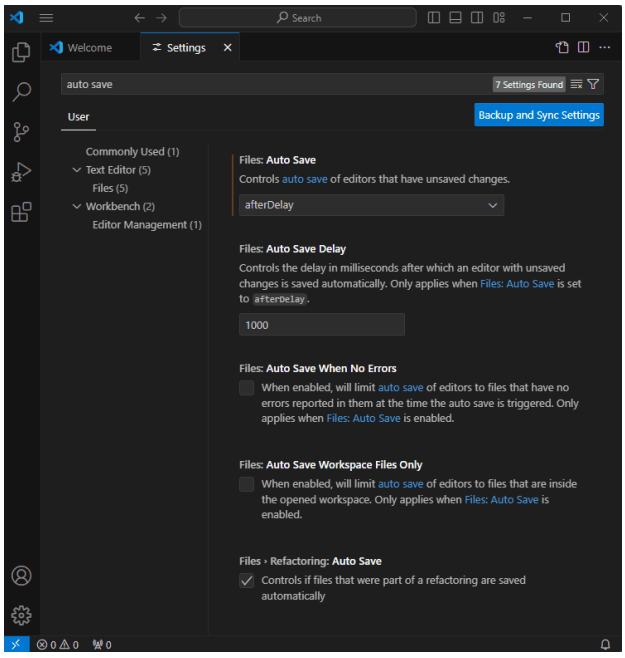
Sidebar in VS Code has , Explorer, Search, Source Control, Run & Debug, Extensions Accounts, and Manage Which we will use frequently as we proceed further



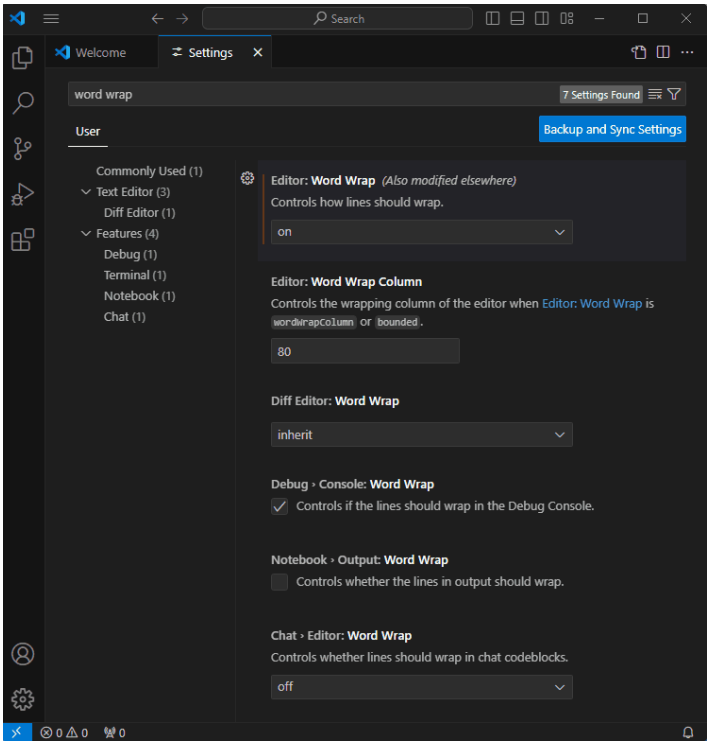
Auto save (afterDelay)



Word wrap



Format on Paste & Save



Git & GitHub

Git for windows : [Click here]

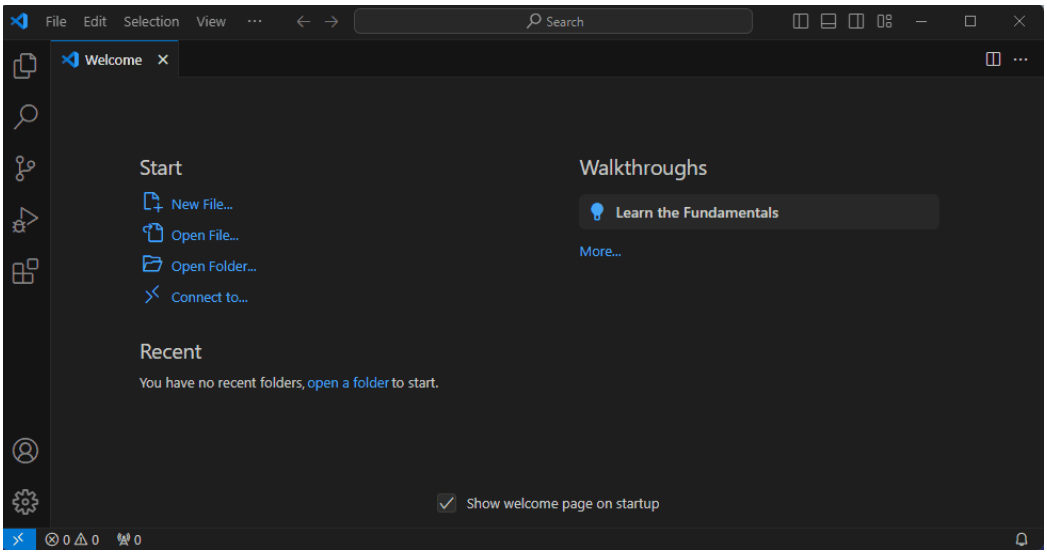
What is Git?

Git is a tool that helps you track changes to your code. Imagine writing a long essay and wanting to save different versions of it as you make changes. Git does this for your code, allowing you to:

- Save Snapshots: Take snapshots of your code at different points (called commits).
- Track Changes: See what changes were made and who made them.
- Revert Changes: Go back to previous versions if something goes wrong.

Download Git

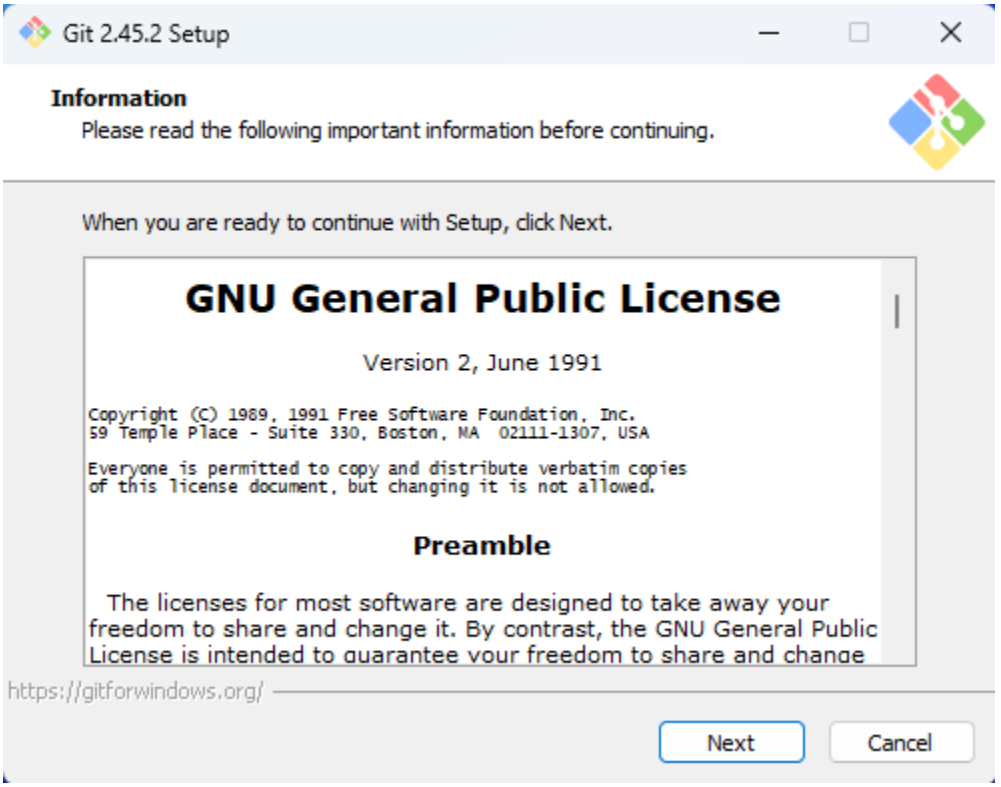
Click on Source Control > Click on Download Git for Windows > Click on allow > Click on 32 bit or 64 bit Git for Windows Setup as per the System



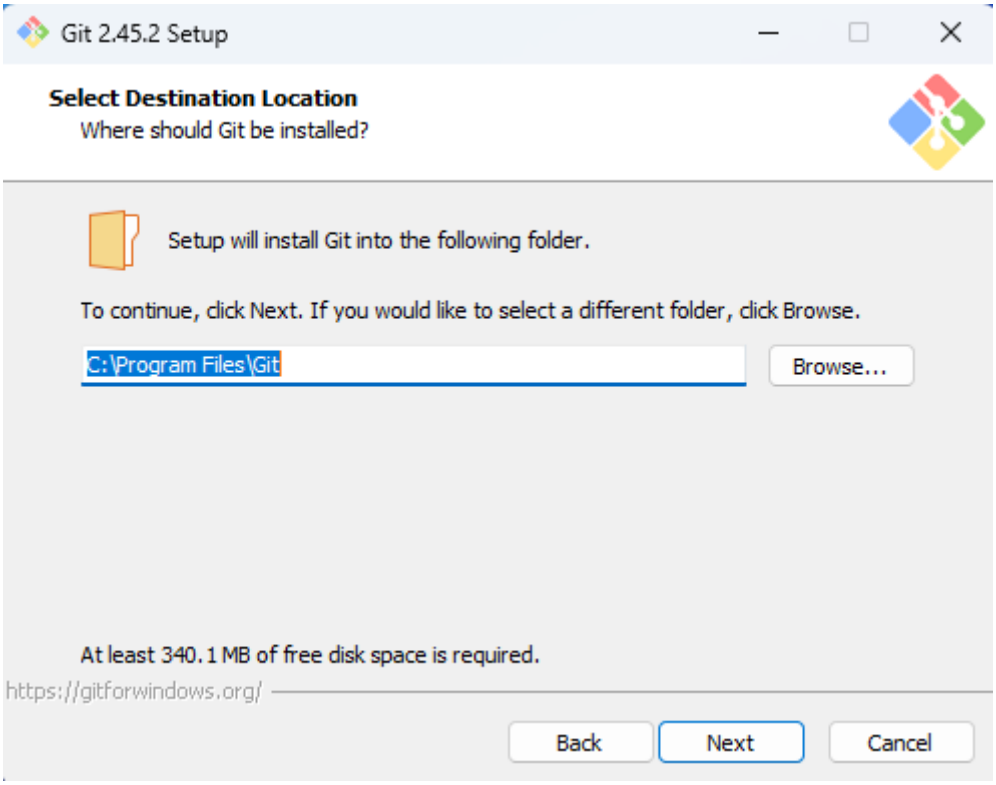
NOTE : Make sure you close the Visual Studio Code before you start installation of Git

Install Git

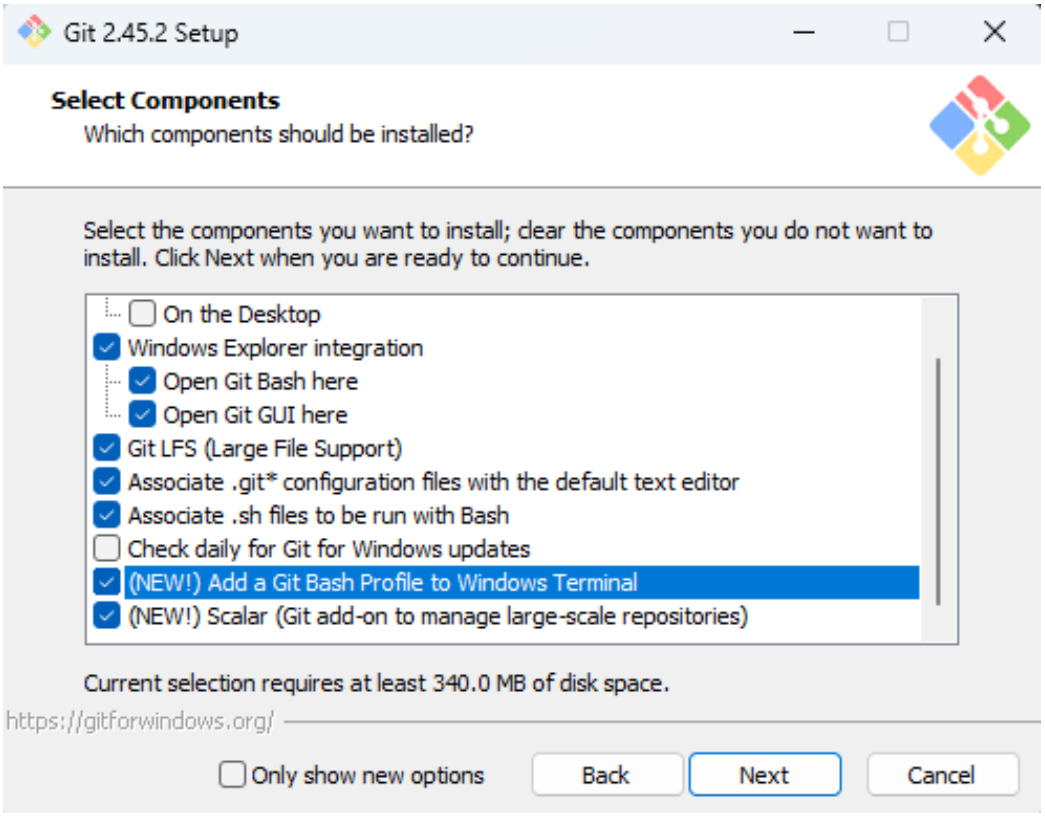
Step 1 : Allow the .exe file to run on your system and click on next



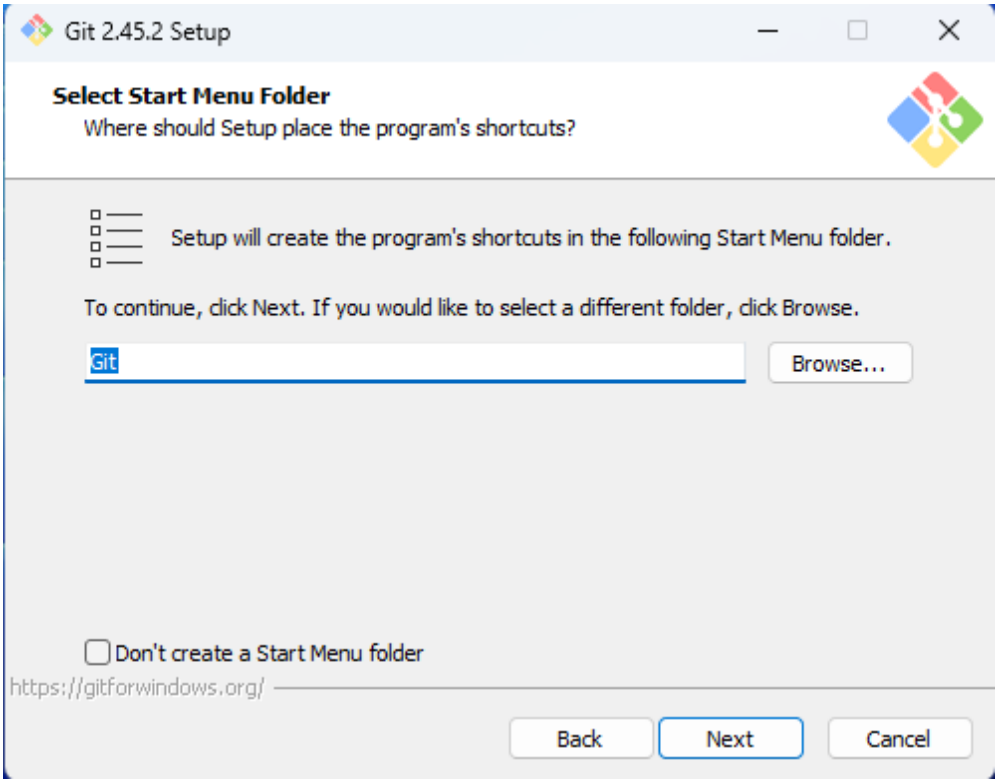
Step 2 : Click on next



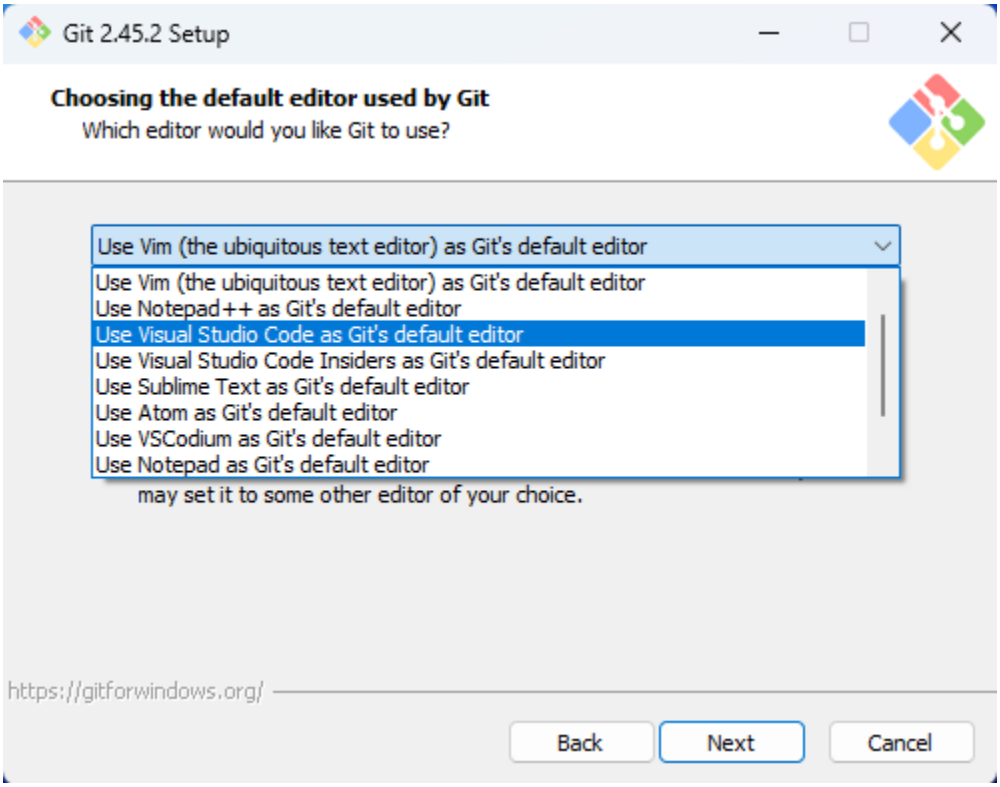
Step 3 : Scroll down & select (NEW!) Add a Git Bash Profile to Windows Terminal



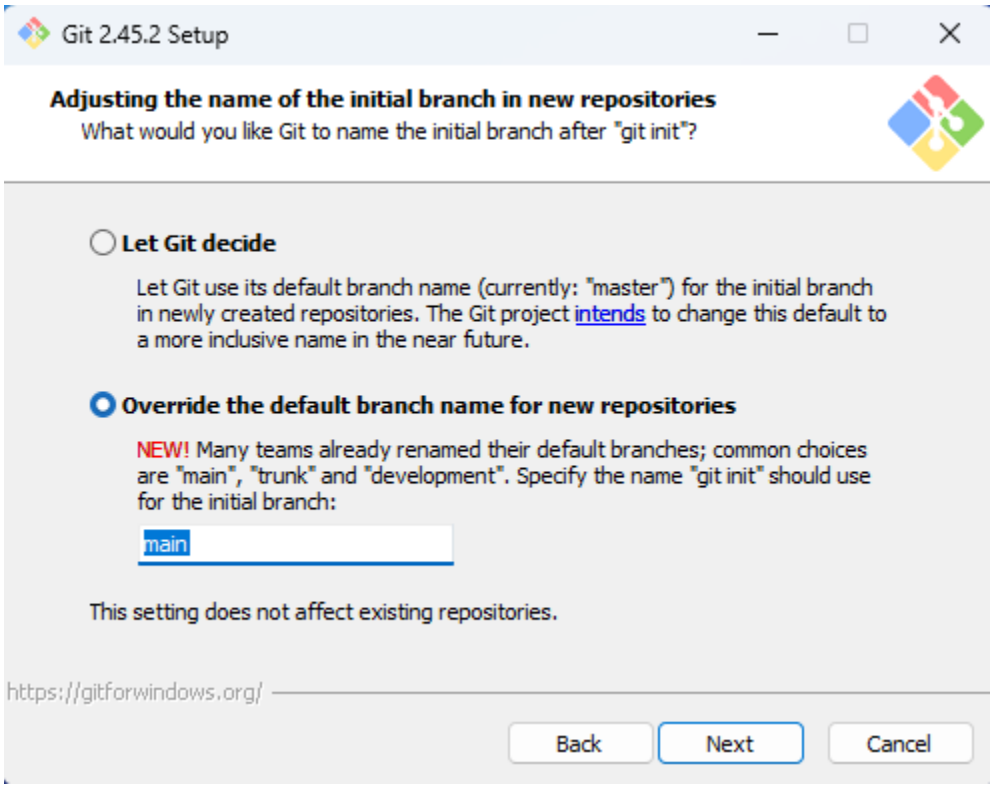
Step 4 : Click on next



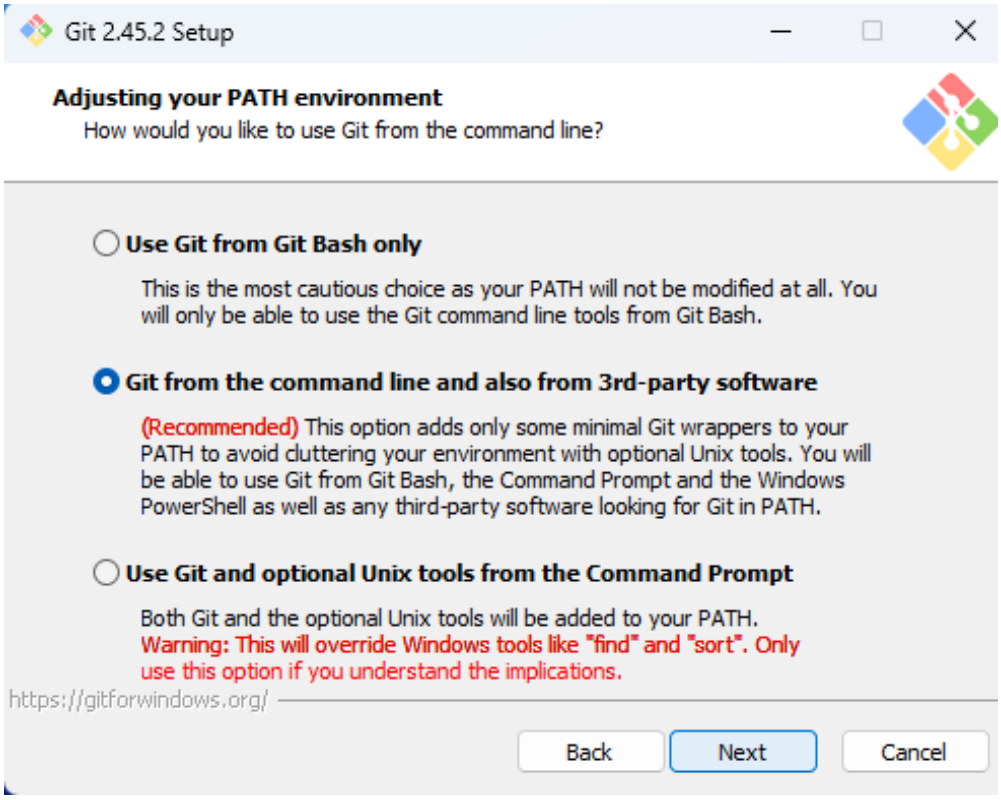
Step 5 : Select Use Visual Studio Code as Gits default editor & click on next



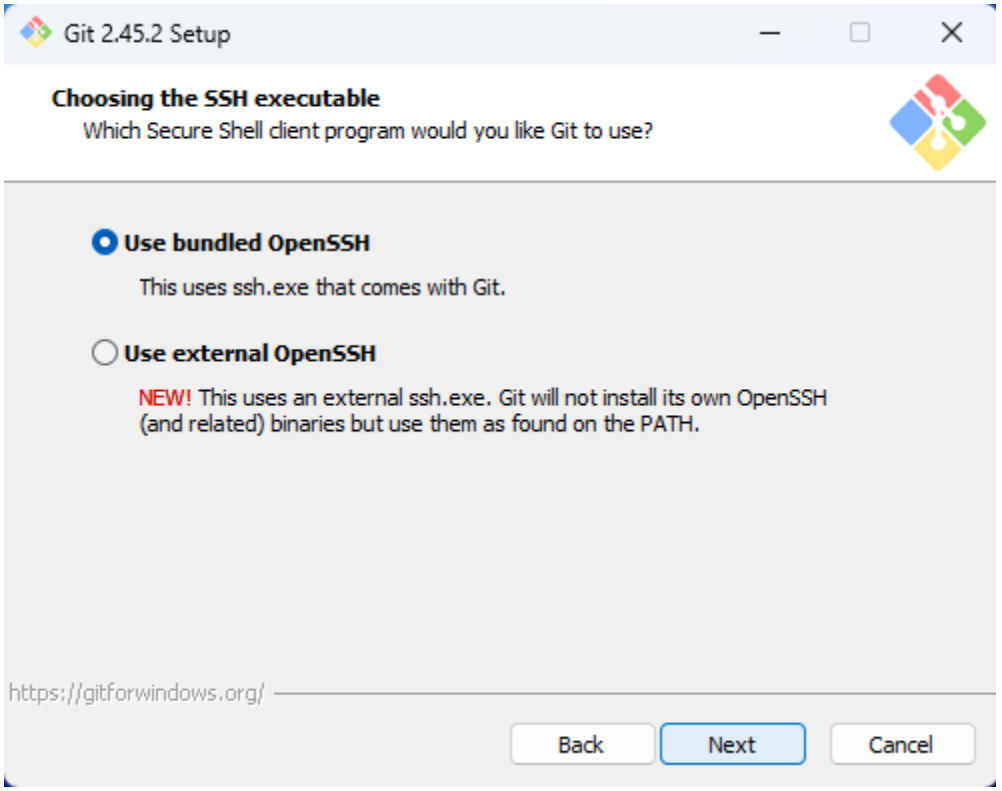
Step 6 : Select Override the default branch name for new repositories & Click on next



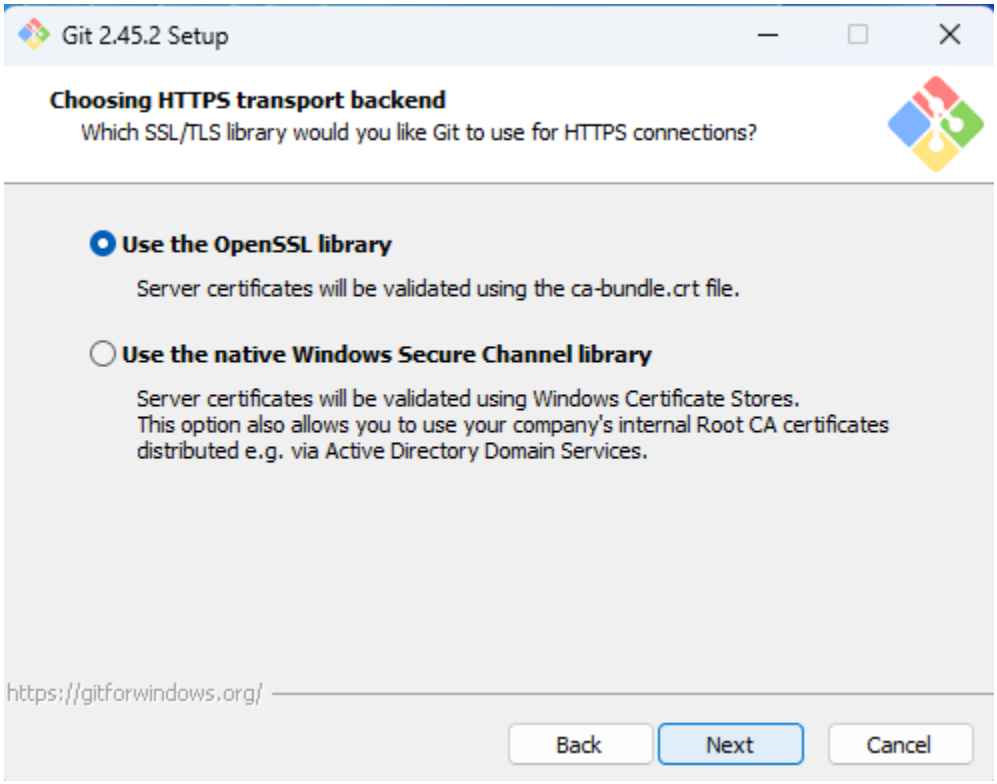
Step 7 : Match the screenshot given below & Click on next.



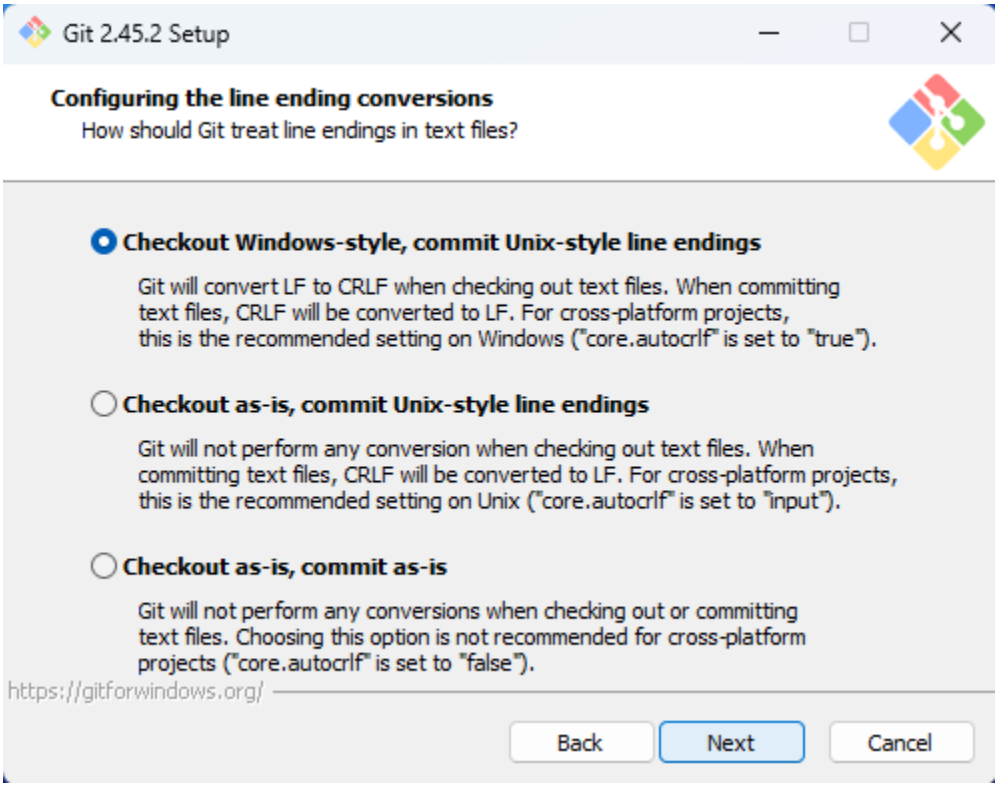
Step 8 : Match the screenshot given below & Click on next



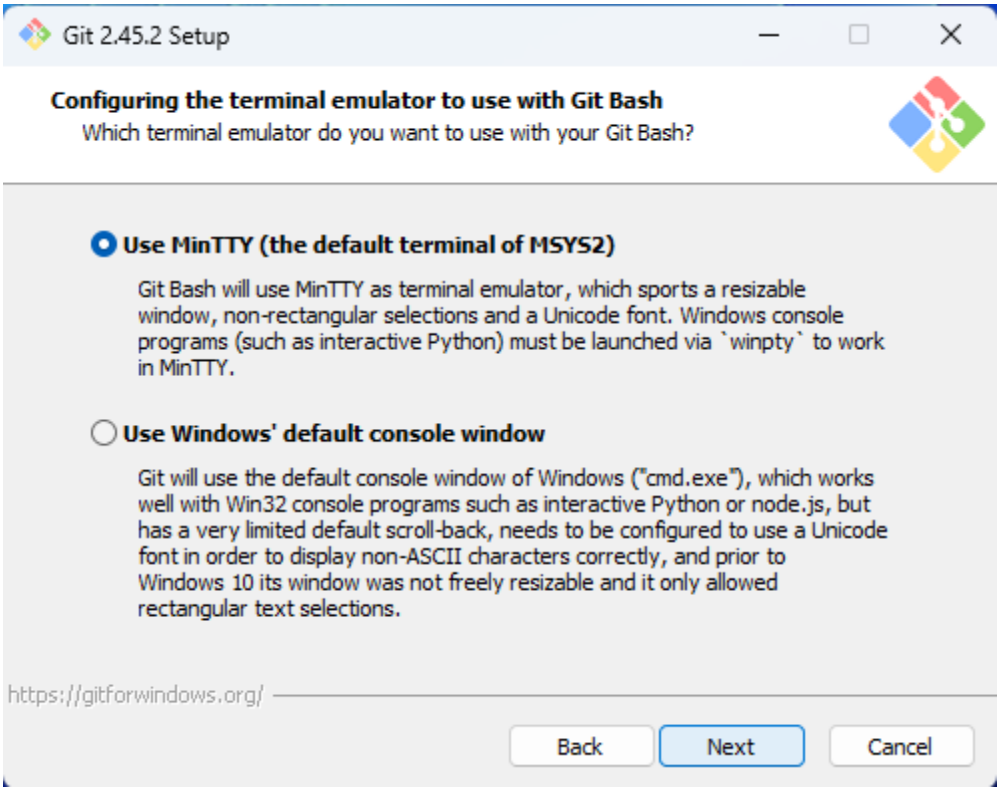
Step 9 : Match the screenshot given below & Click on next



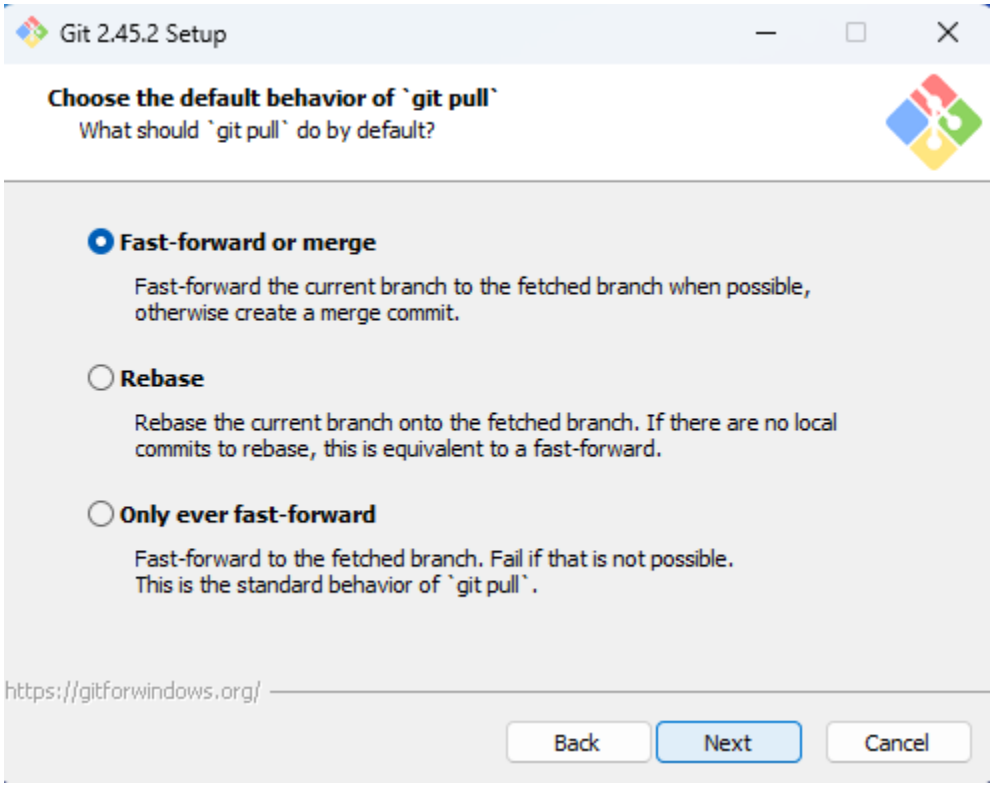
Step 10 : Match the screenshot given below & Click on next



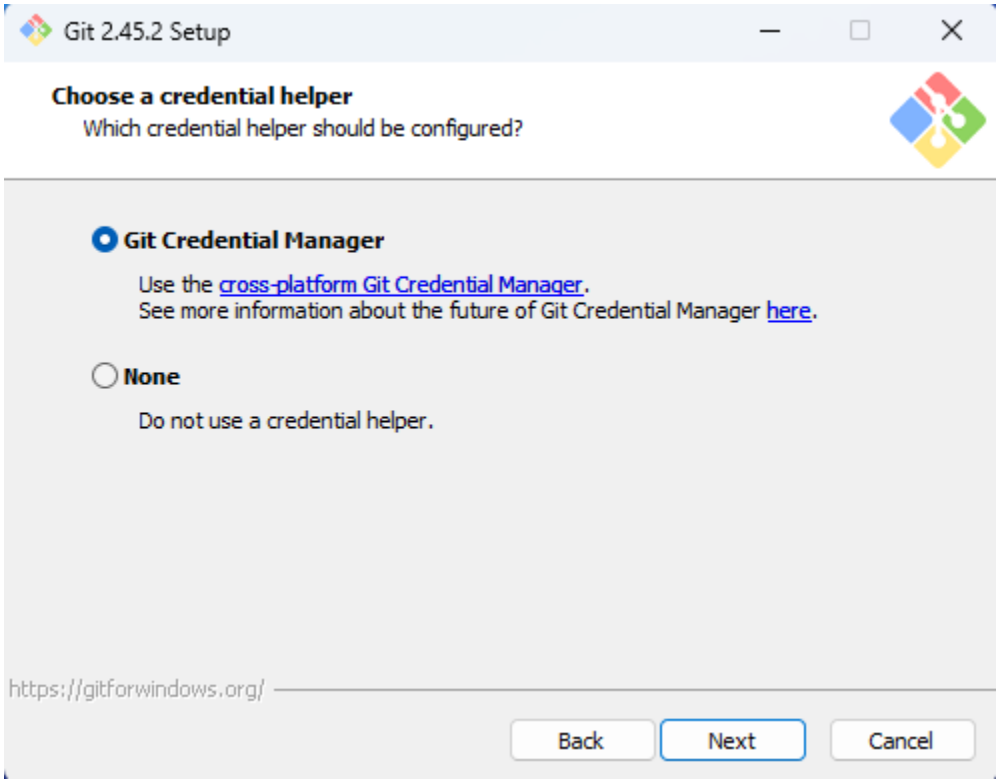
Step 11 : Match the screenshot given below & Click on next



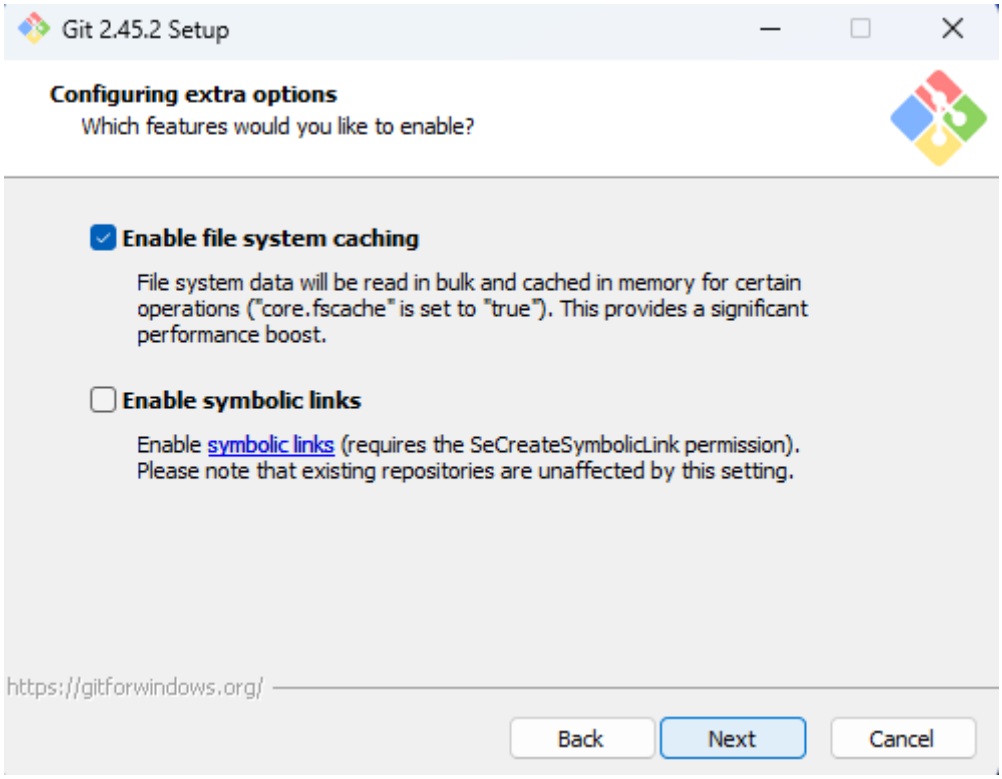
Step 12 : Match the screenshot given below & Click on next



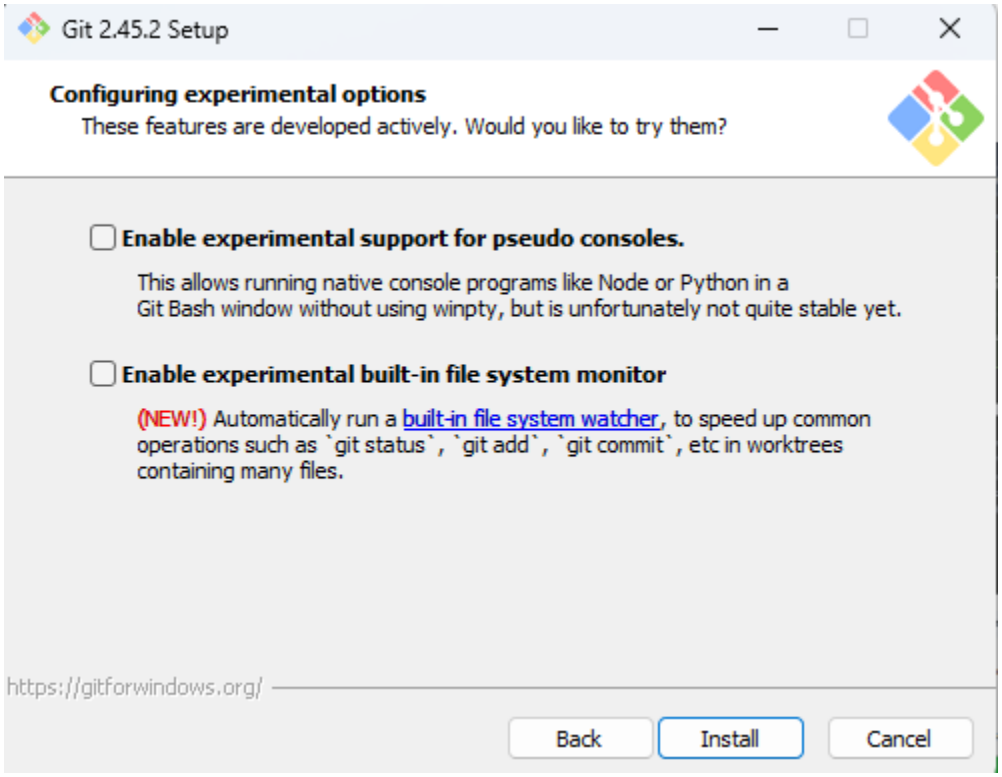
Step 13 : Match the screenshot given below & Click on next



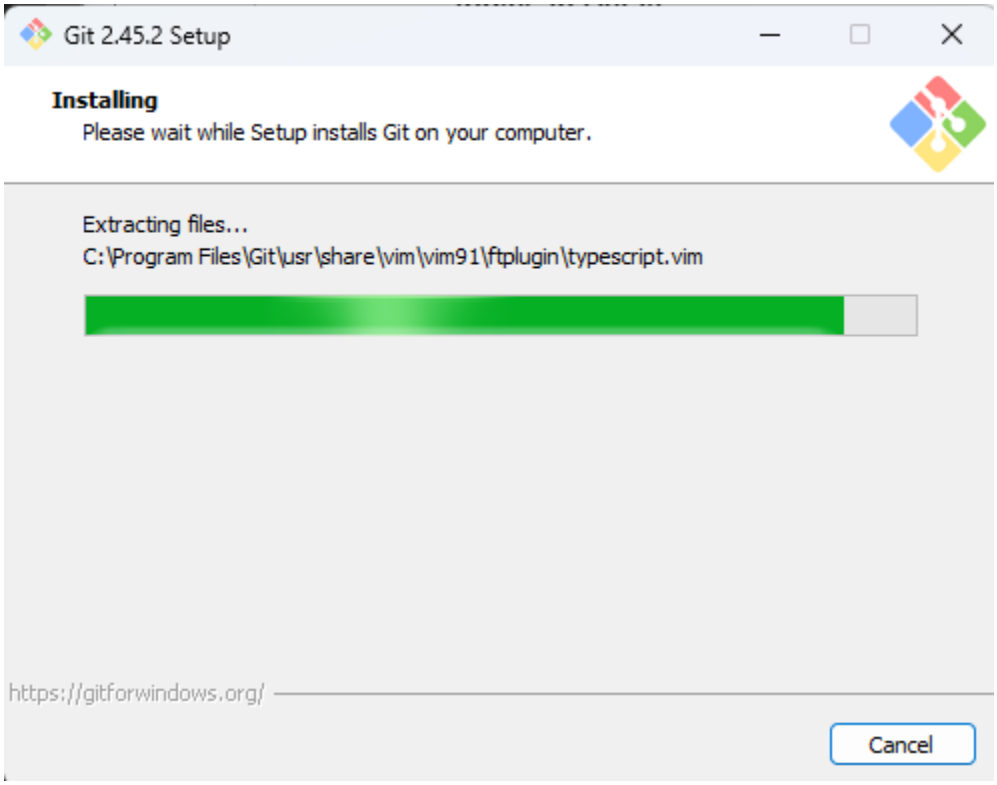
Step 14 : Match the screenshot given below & Click on next



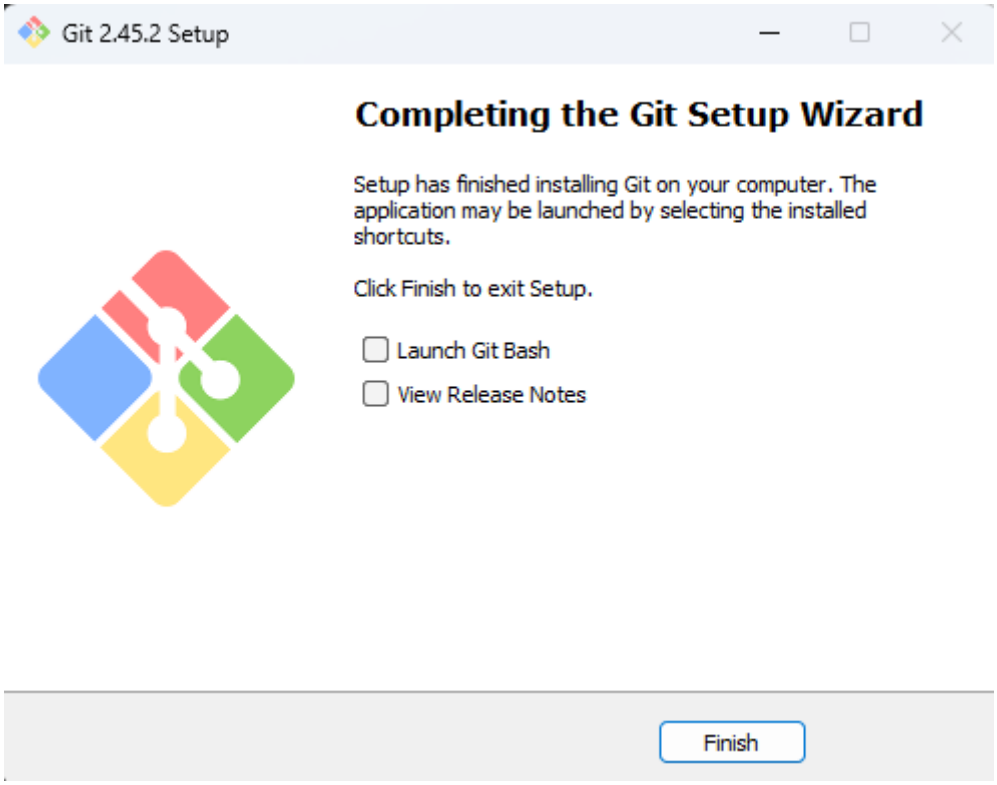
Step 15 : Match the screenshot given below & Click on install



Step 16: Wait for the installation to complete



Step 17 : Match the screenshot given below & Click on Finish



Step-by-Step Guide to Install Git on a Mac

Using Terminal (Command Line Interface)

- Open Terminal:
  - Click on the Spotlight icon (magnifying glass) in the upper-right corner of your screen.
  - Type "Terminal" and press Enter to open it.
- Install Xcode Command Line Tools:
  - In the Terminal window, type the following command and press Enter : xcode-select --install
- Follow the On-Screen Instructions:
  - A pop-up window will appear asking if you want to install the command line tools. Click on "Install" to proceed.
  - Agree to the license agreement if prompted.
- Wait for the Installation to Complete:
  - The installation process will take a few minutes. Once it's done, you can close the pop-up window.
- Verify the Installation:
  - In the Terminal window, type the following command and press Enter to check if Git is installed: git --version
  - You should see a version number displayed, indicating that Git is successfully installed.

What is GitHub?

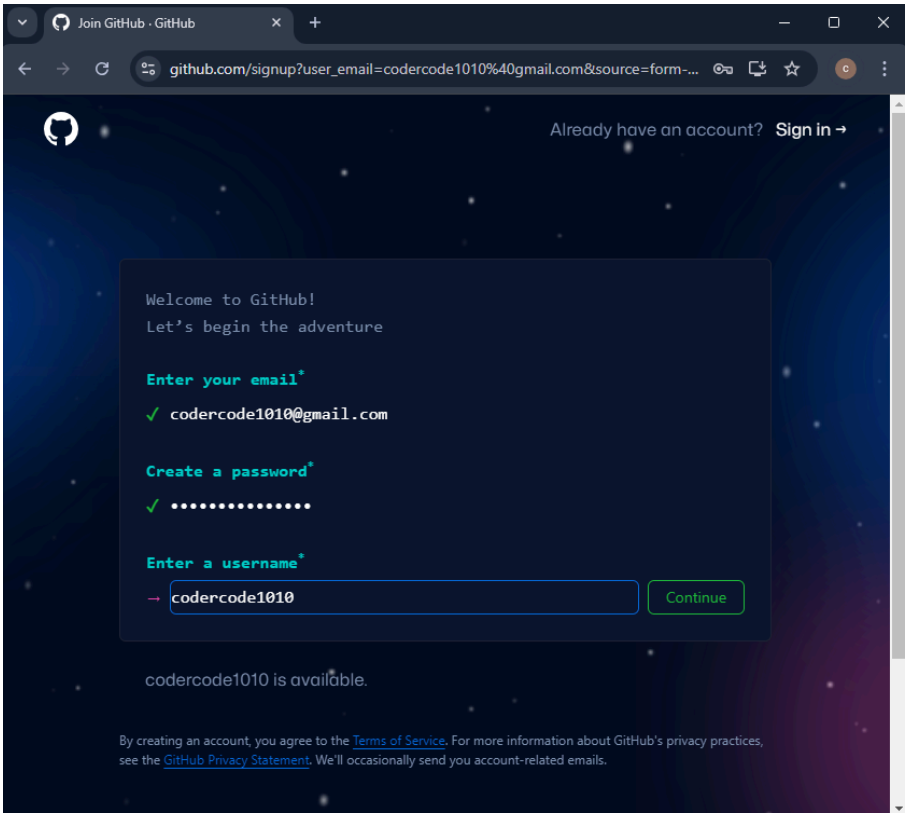
GitHub is a website that lets you store and share your Git projects online. It's like a social media platform for code, where you can:

- Collaborate: Work with others on the same project, even if you're in different places.
- Share: Make your projects public so anyone can see them, or keep them private.
- Contribute: Help improve other people's projects by suggesting changes.

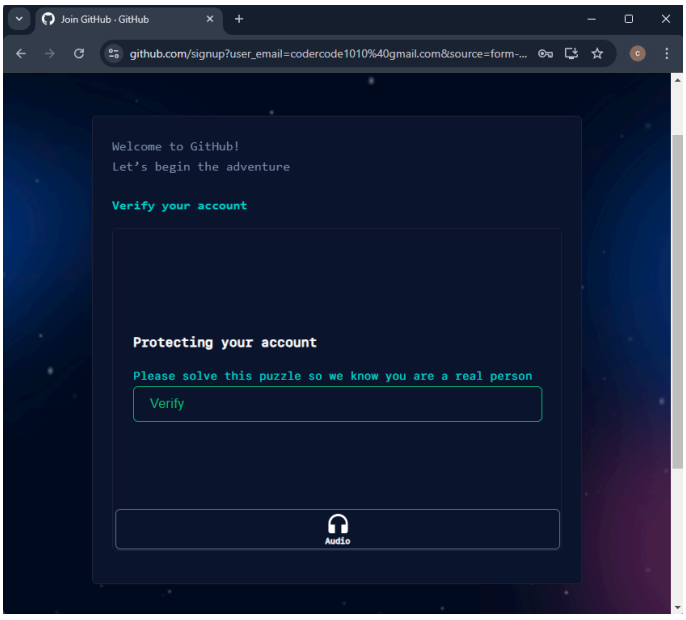
Together, Git and GitHub make it easy to manage your code, work with others, and share your projects with the world.

Creating GitHub Account

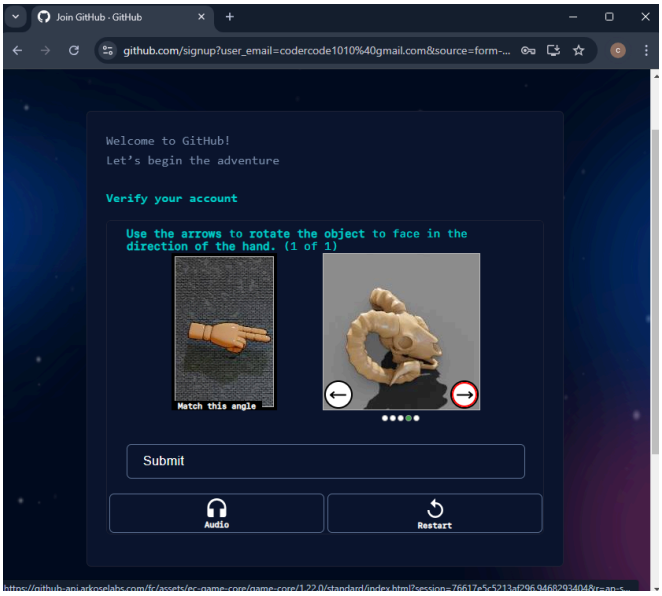
Step 1 : Enter students email, Create password & username



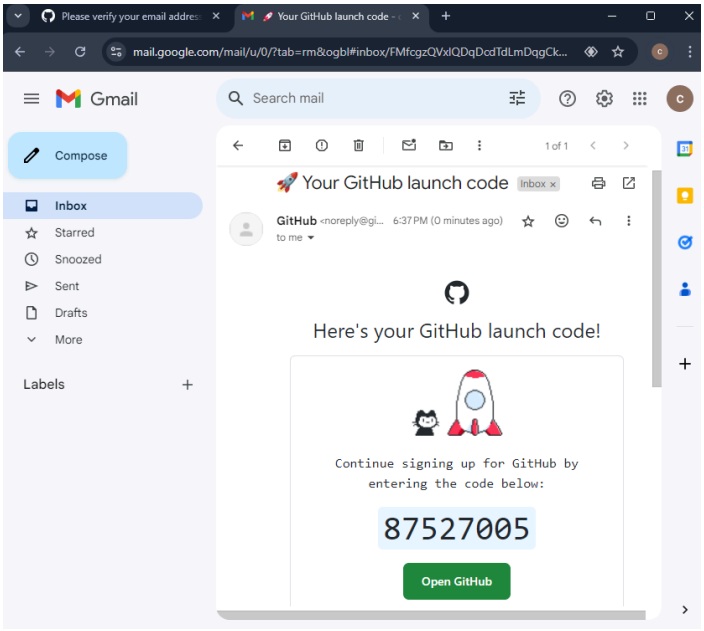
Step 2 : Click on Verify



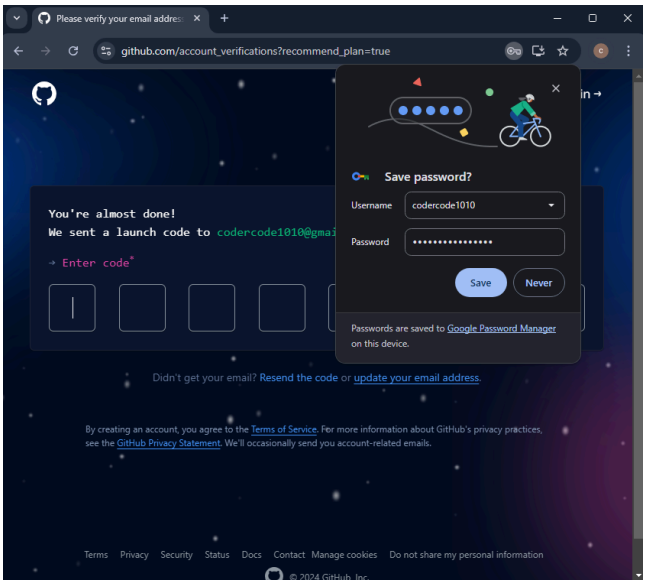
Step 3 : Complete the verification & click on submit



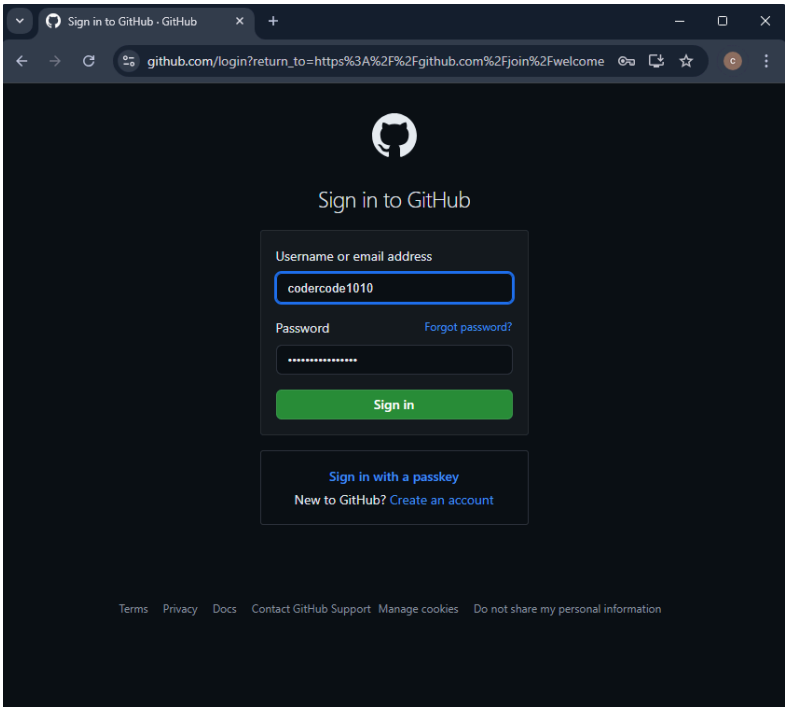
Step 4: Copy the email verification code that is sent on the respective email id



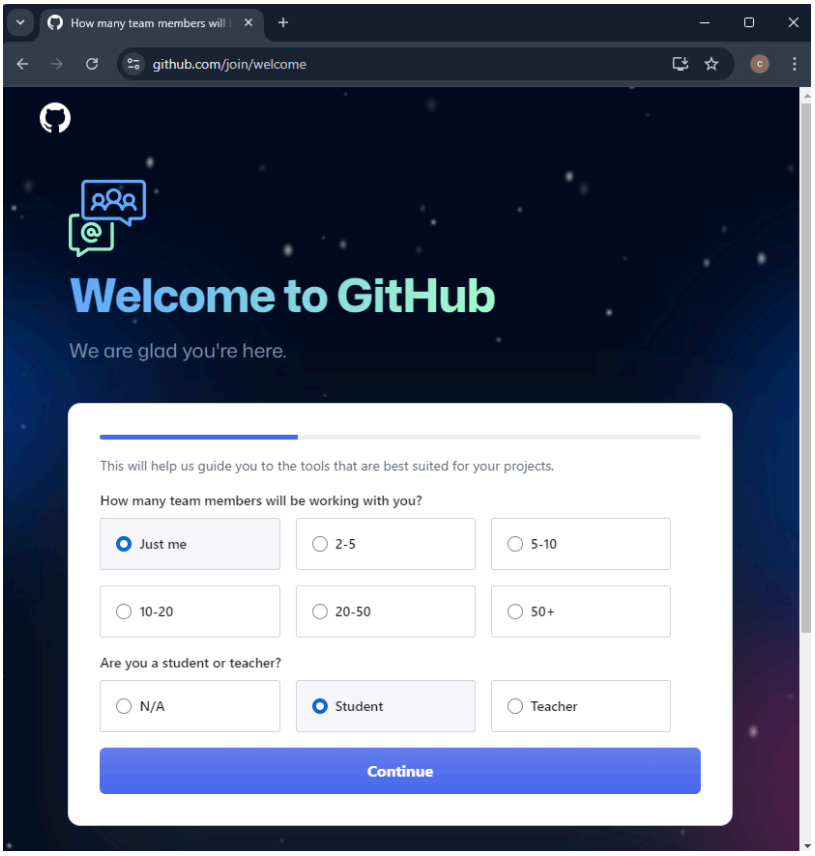
Step 5 : click on save password & paste the email verification code



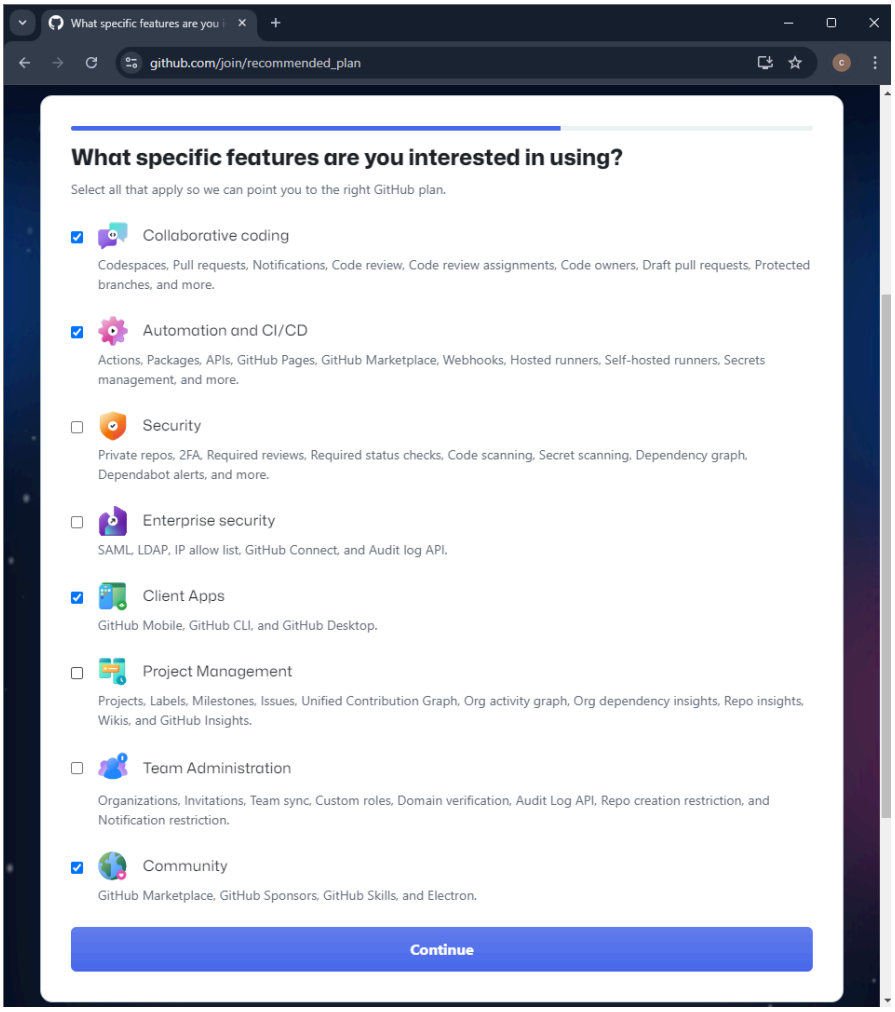
Step 6 : Click on Sign in



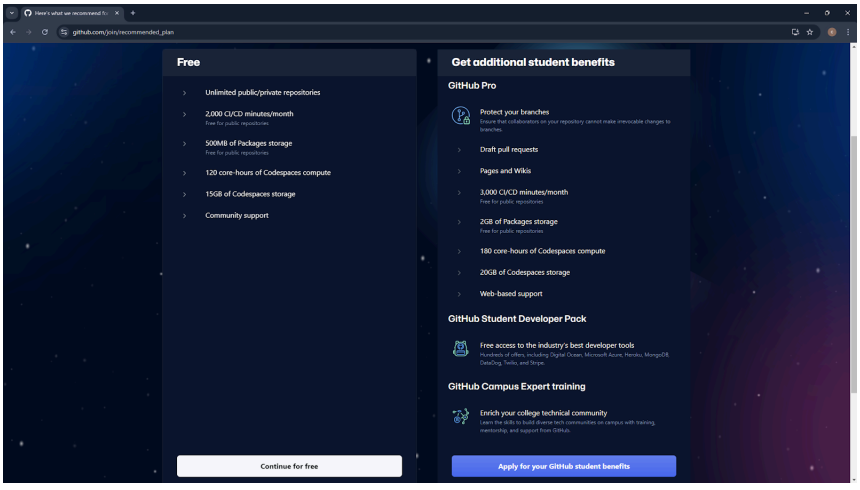
Step 7 : select Just me , Student & click on continue



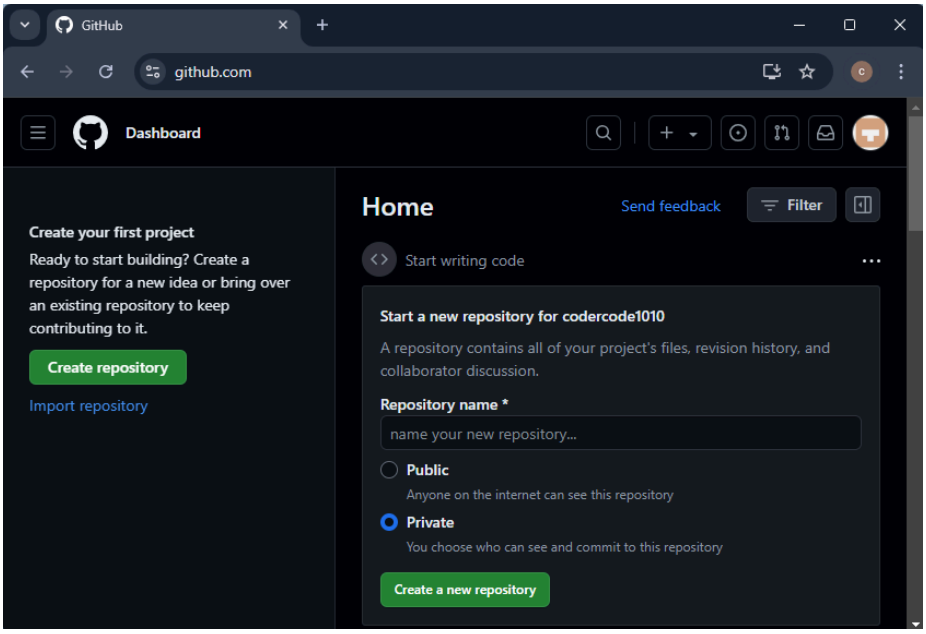
Step 8 : Select any 1 & Click on continue



Step 9 : Click on continue for Free



Step 10 : Congratulations your GitHub account is ready



Folder Structure

What is a Folder Structure?

A folder structure is a way to organize files on your computer. It helps keep your work neat and easy to find.

Why is it Important?

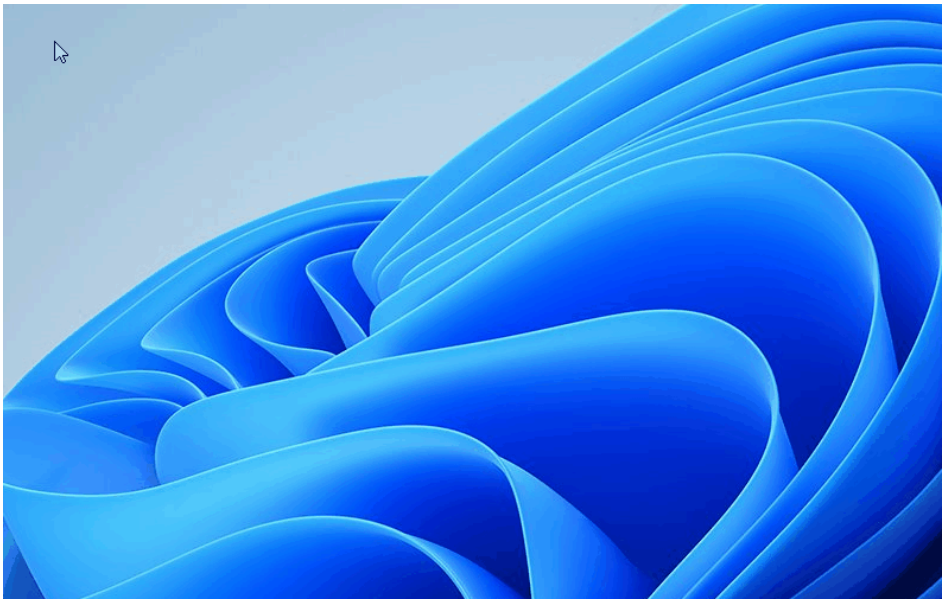
Organization: Keeps your files organized so you can easily find what you need.

Clarity: Makes it clear where each file belongs, especially when working on multiple projects.

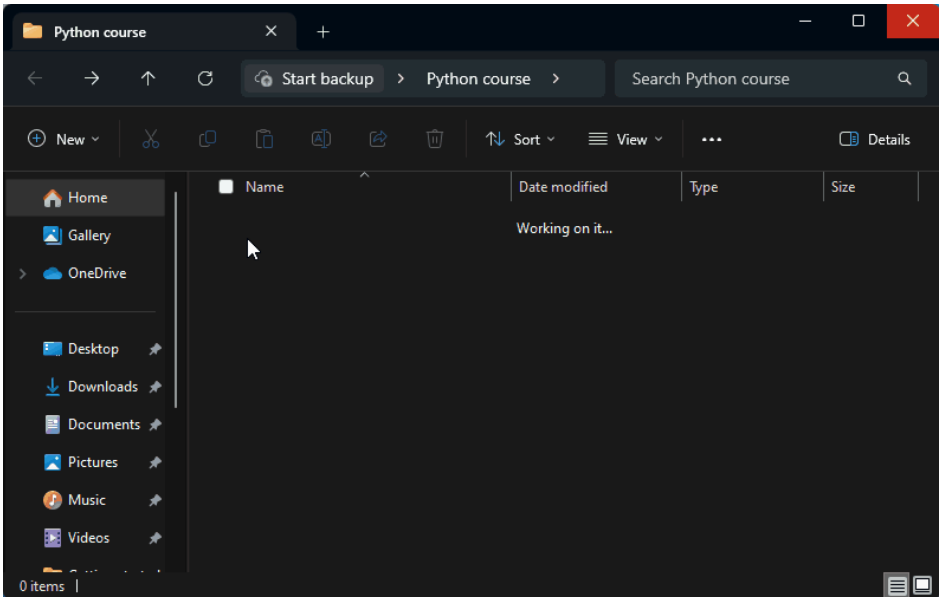
Efficiency: Saves time by avoiding clutter and confusion.

Our Folder Structure

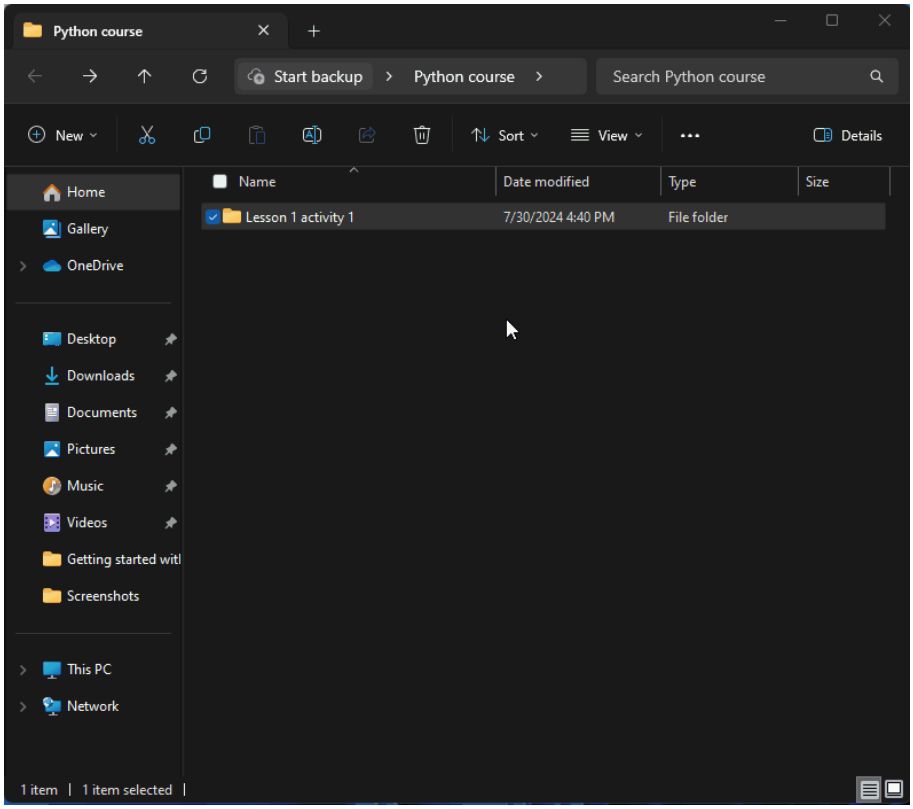
Step 1 : Create a Folder Named after your course & open it as shown in the example below



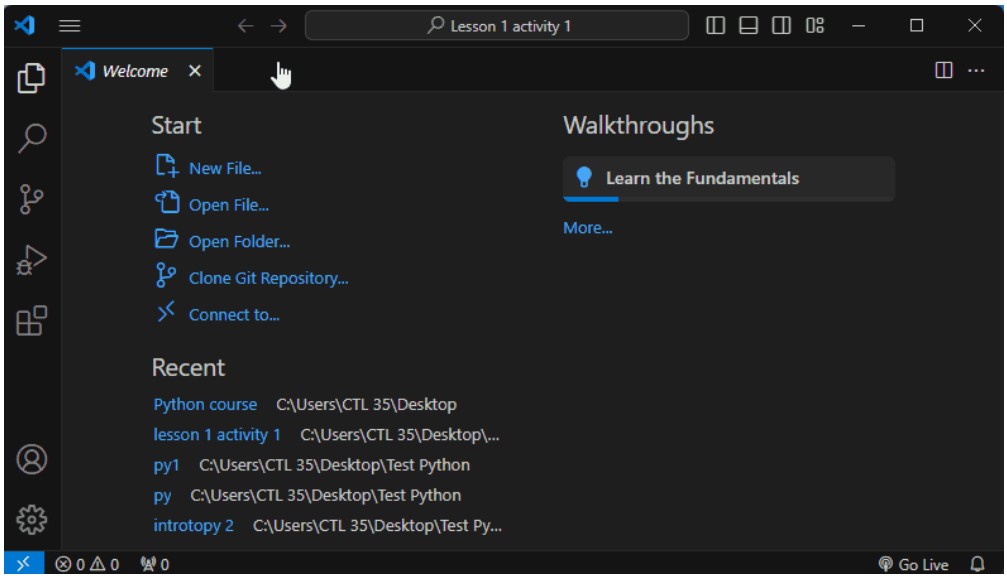
Step 2 : Create a Folder Named after Lesson & activity as shown



Step 3 : Open the Folder using VS Code

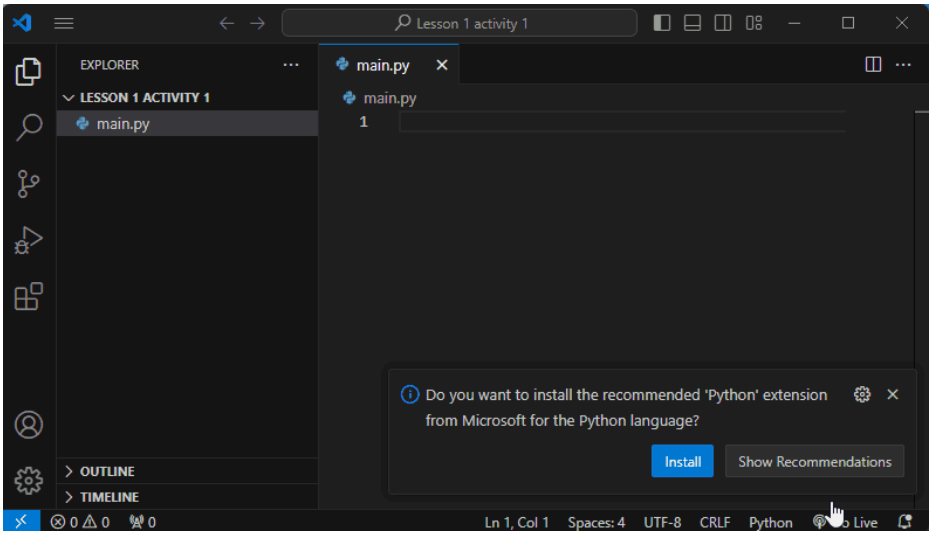


Step 4: Create main.py file

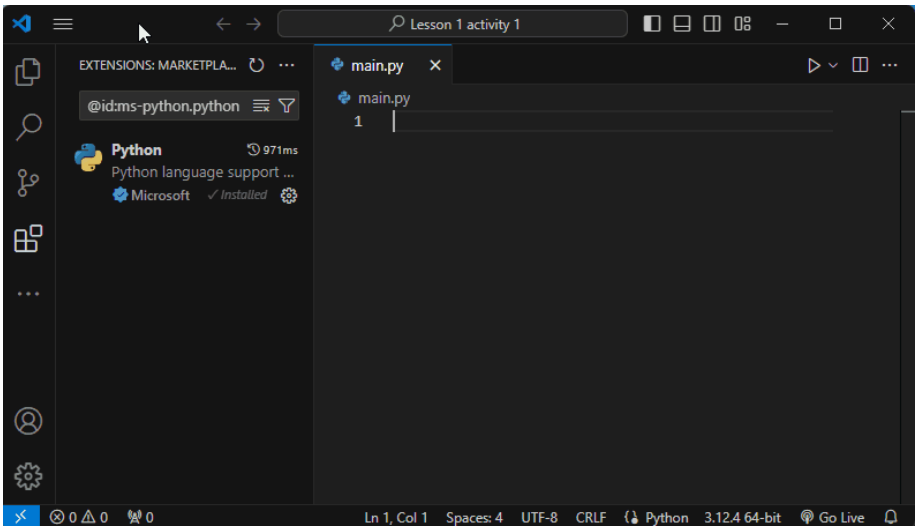


Install & Use extensions in VS Code

Install Python Extension



Run Python



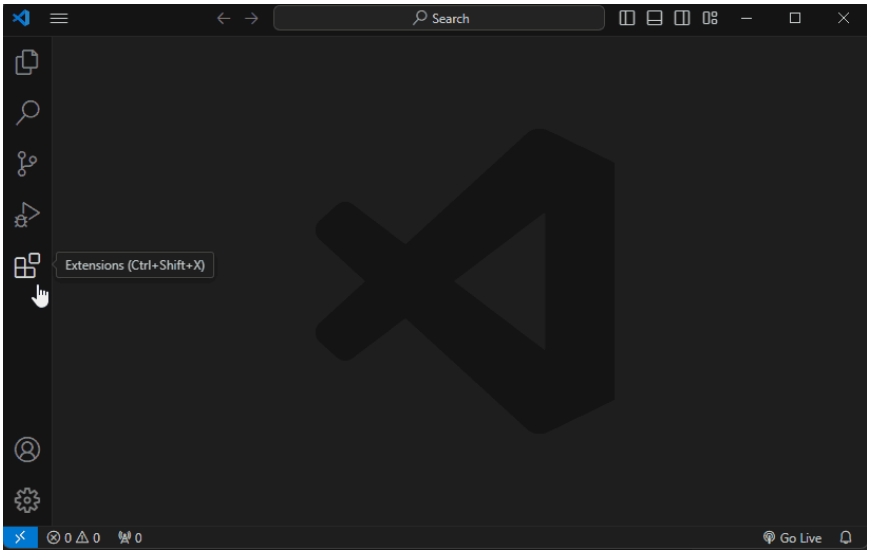
Note: You can also run the file in the terminal using the following command, and it will accept user input.

macOS: Run python3 filename.py in the VS Code terminal, replacing filename.py with the name of your Python file.

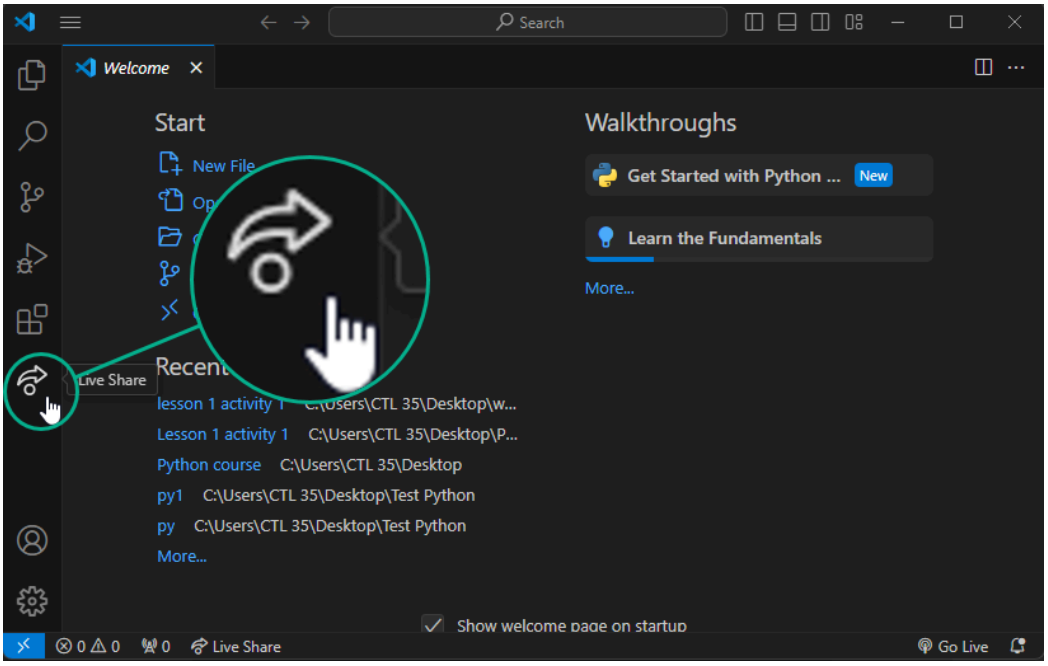
Windows: Run python filename.py in the VS Code terminal, replacing filename.py with the name of your Python file.

Install Live Share

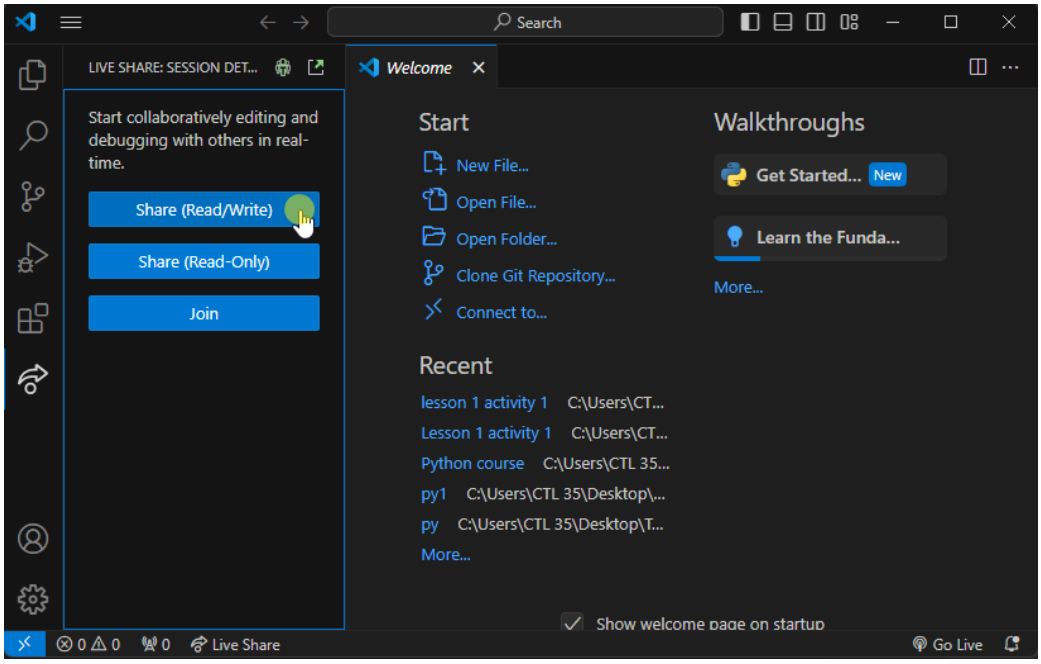
NOTE : This extension enables teachers to join students' Visual Studio Code editor, allowing them to provide real-time guidance as students code along.



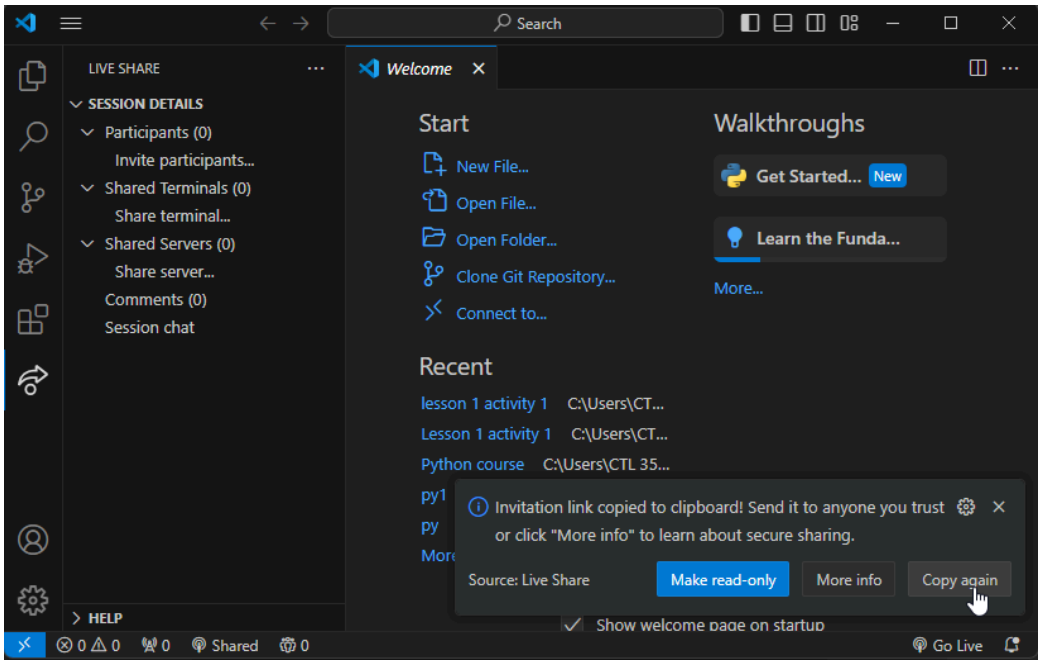
Click on Live Share



Click on Share(Read/Write) & wait for the Link to generate

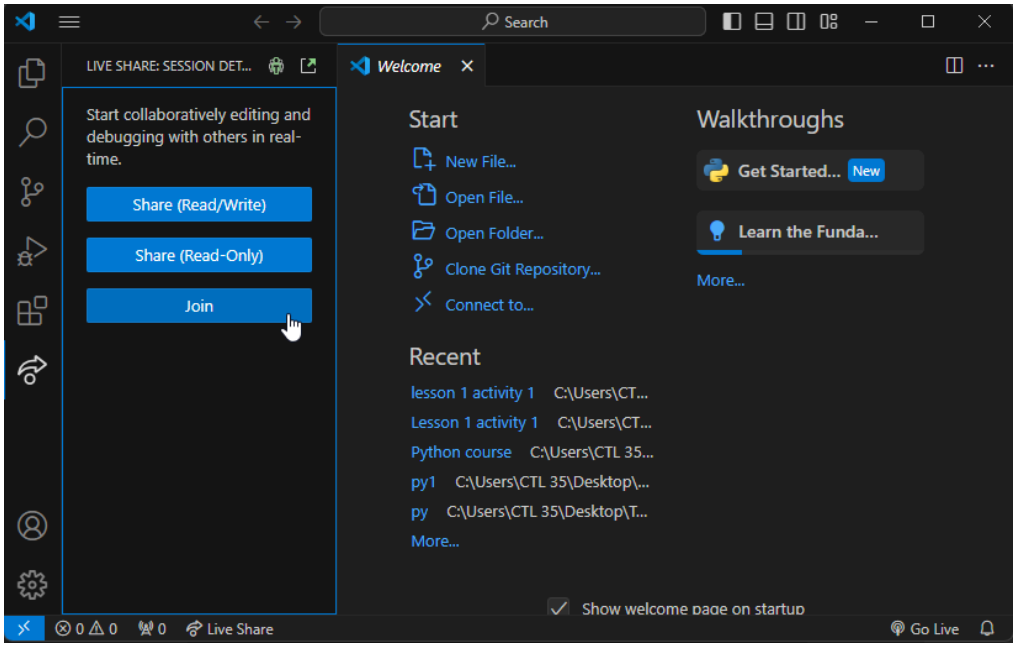


You will get a Notification of Link has been copied to your Clipboard > Go back to the Classroom > click on chat > paste the Link for teachers to Join

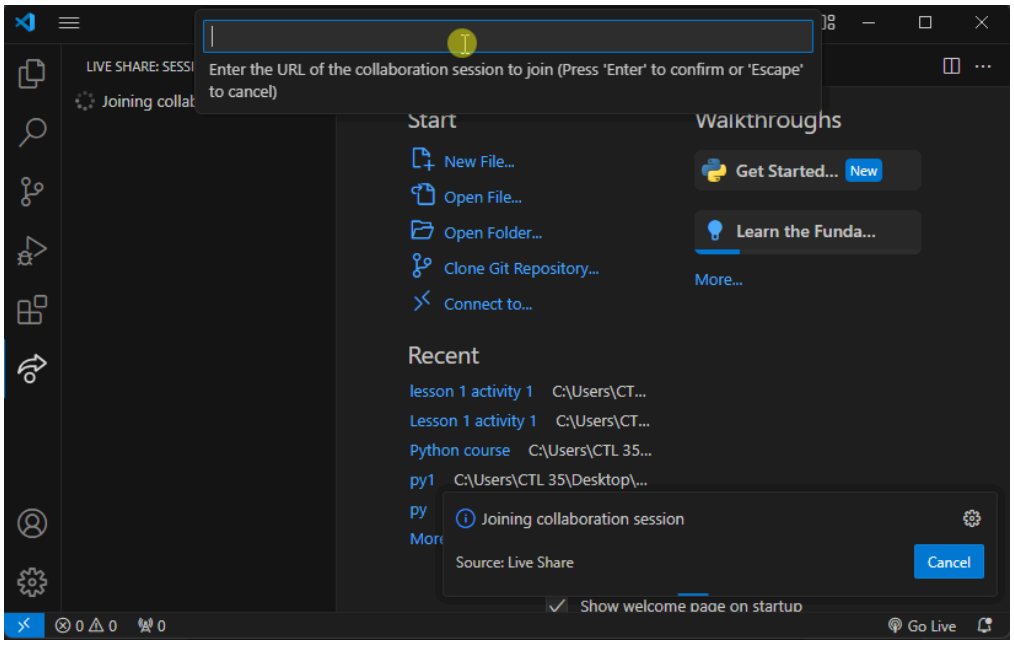


Teachers Can copy the Join Link

Click on Join



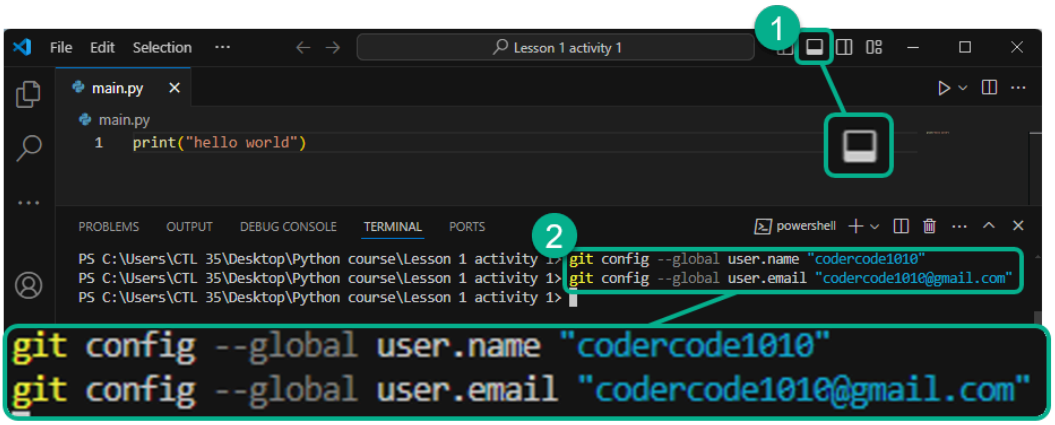
Paste the Url



Configure username & email id using git config

NOTE : This step is mandatory and should not be skipped. Use the highlighted screenshots provided to complete the configuration.

Open the terminal in VS Code as shown



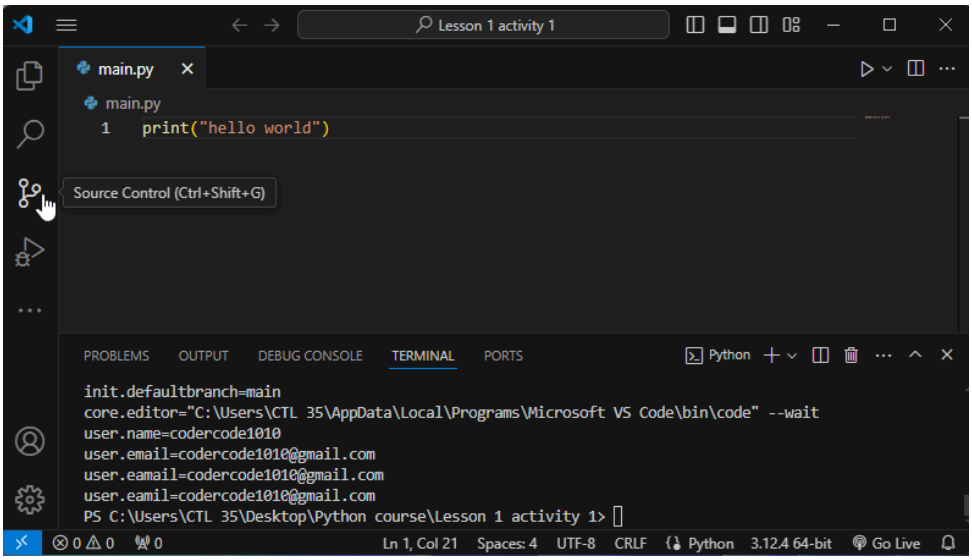
Step 1 : Click on the Terminal icon as shown to open terminal

Step 2: Type the commands shown followed by Students name & email id

---

Push the code to Github

Step 1 : navigate & click Source control in the sidebar of the vs code as shown

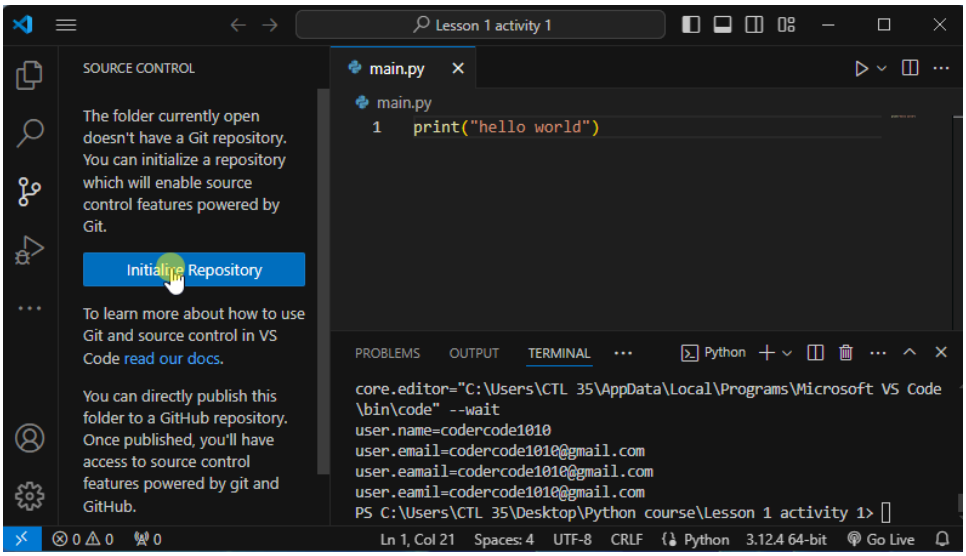


Step 2: Click on initialize repository button as shown in the image

Tell Git to Watch This Folder: When you initialize a repository, you're telling Git to start paying attention to this new folder. Git will then keep track of all the changes you make to the files in this folder.

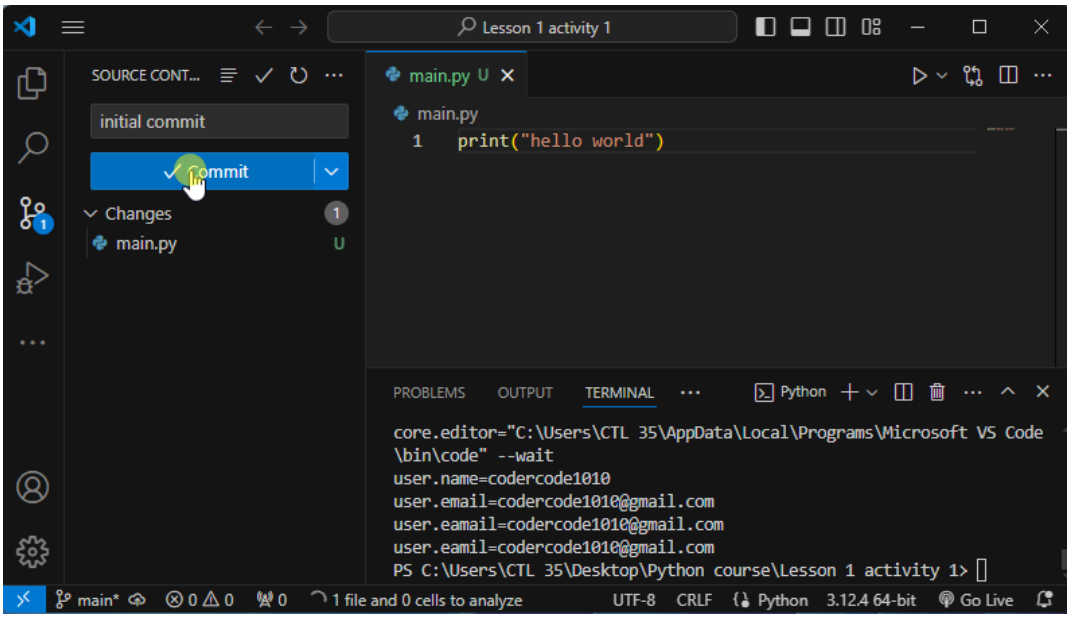
Start Saving Snapshots: Once Git is watching your folder, you can start saving snapshots of your project at different

points. These snapshots are called "commits," and they help you track what changes were made and when.
Track Progress: You can see how your project has changed over time.
Undo Mistakes: If you make a mistake, you can go back to an earlier version of your project.



Step 3 : Write the commit message & click on commit button

NOTE : Be sure to write your commit message first, then click the commit button to save your changes.



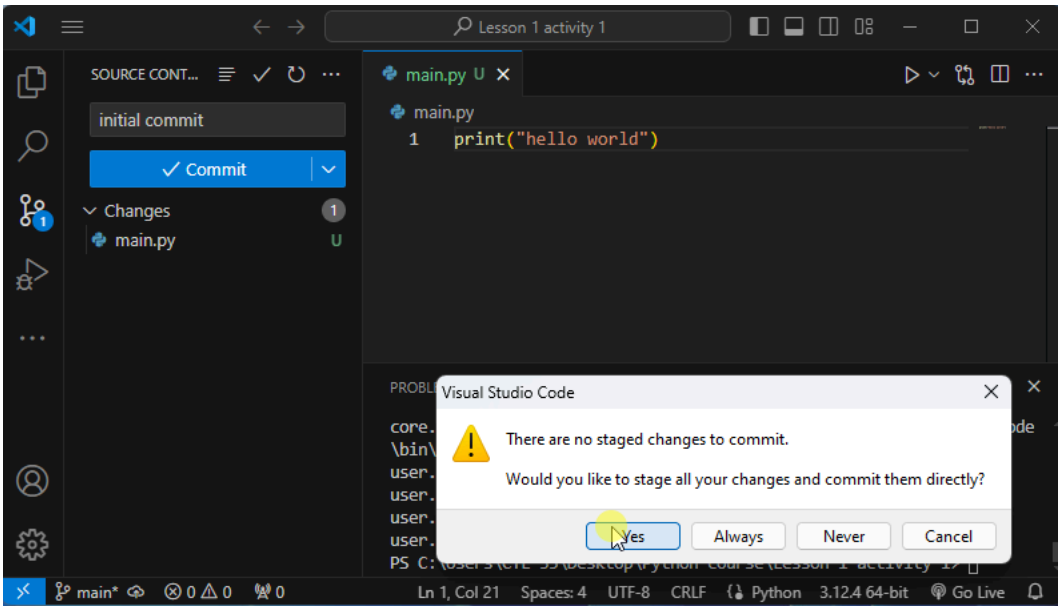
Step 4 : Click on yes to stage all changes

Stage is a way to prepare your changes before saving them in Git. Here's a short and simple explanation:

Make Changes: You write code or make edits in your project.

Stage Changes: You pick which changes you want to save. It's like putting items in a basket before buying them.

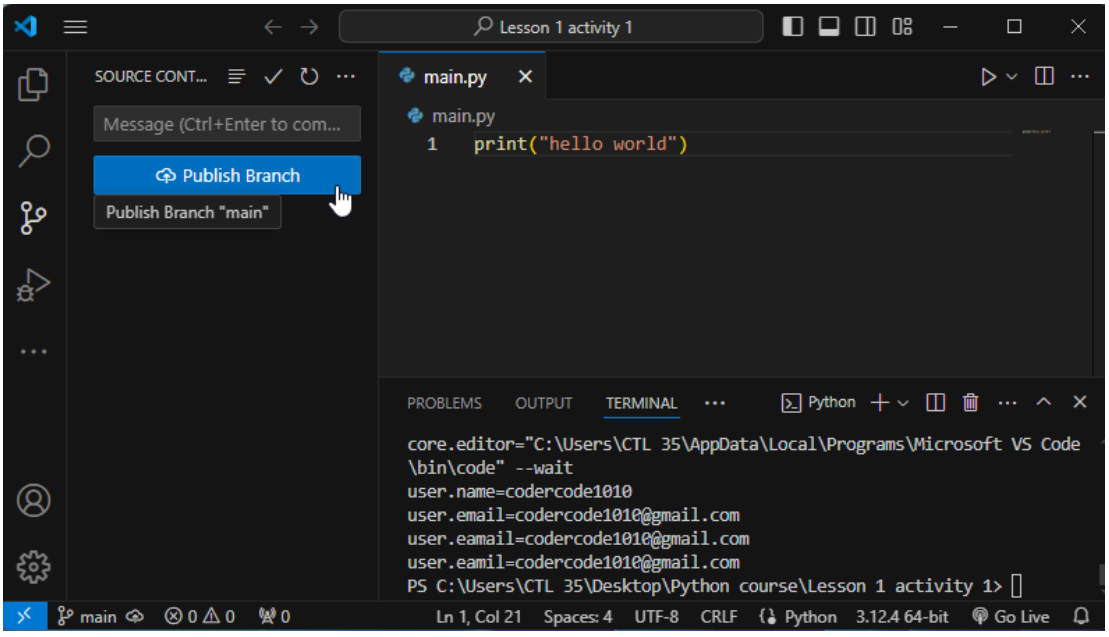
Commit Changes: You save the changes you staged. This creates a snapshot in Git.



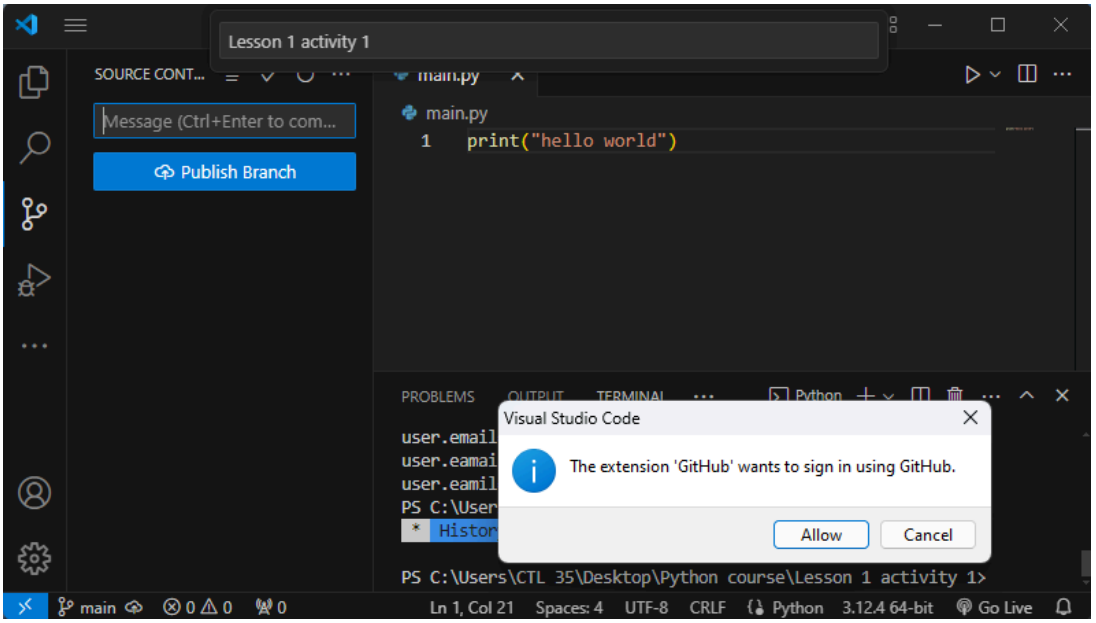
Step 5 : Click on Publish Branch

Publish Branch is about sharing your work with others online, in

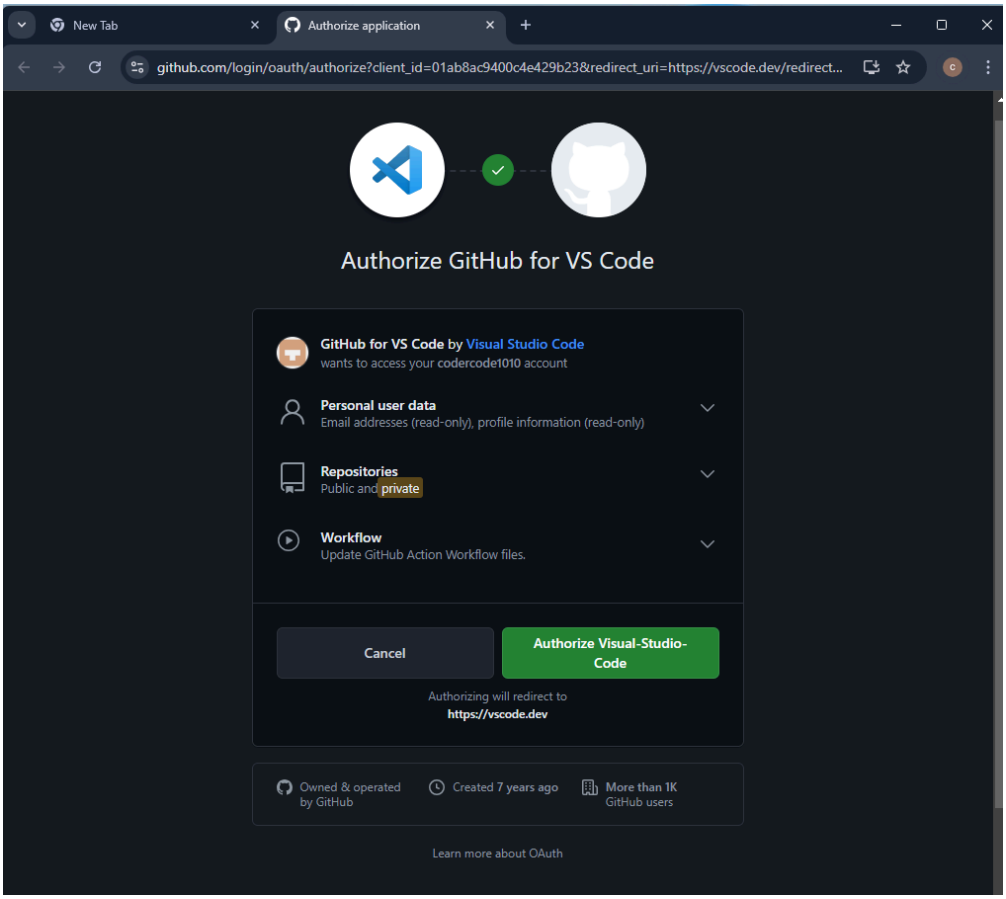
your case so that you can submit in class activities & After Class Project.



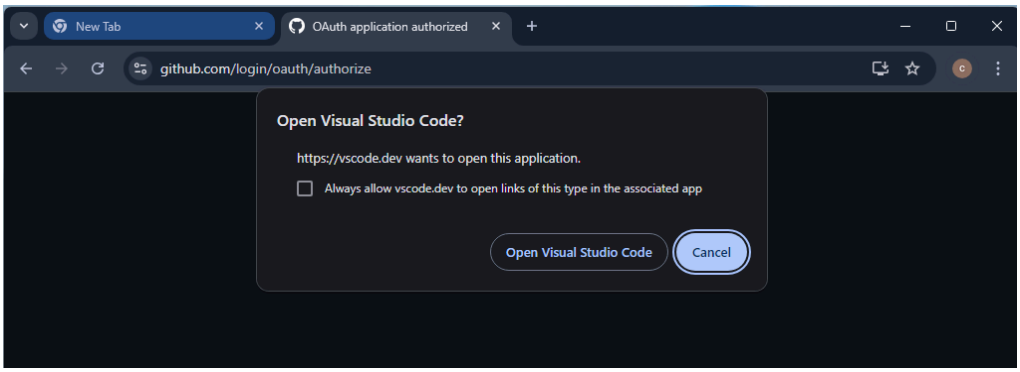
Step 6 : Click on Allow



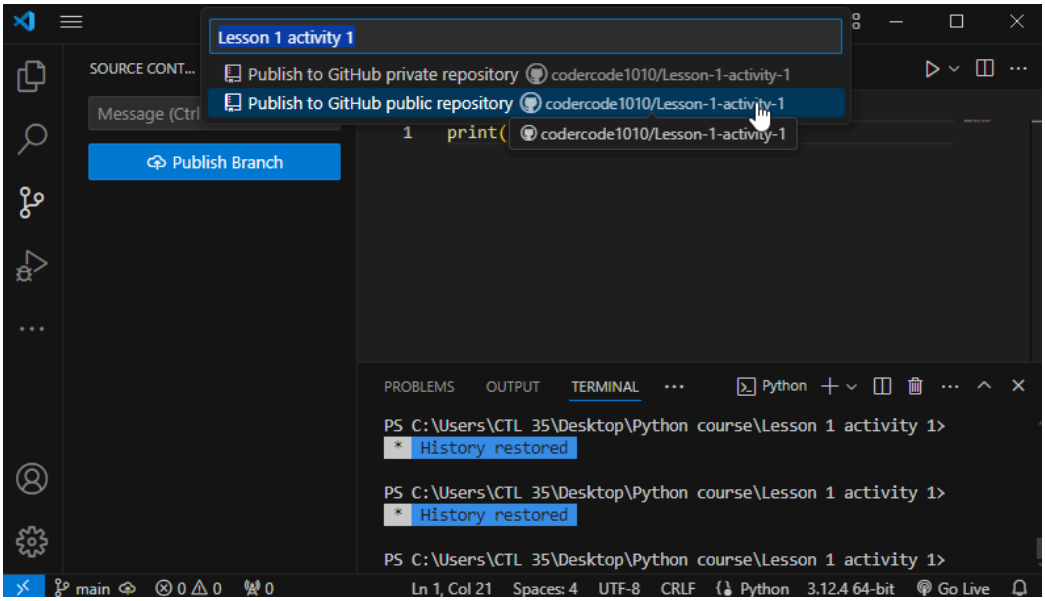
Step 7 : wait for the browser to open & click on Authorize Visual Studio Code



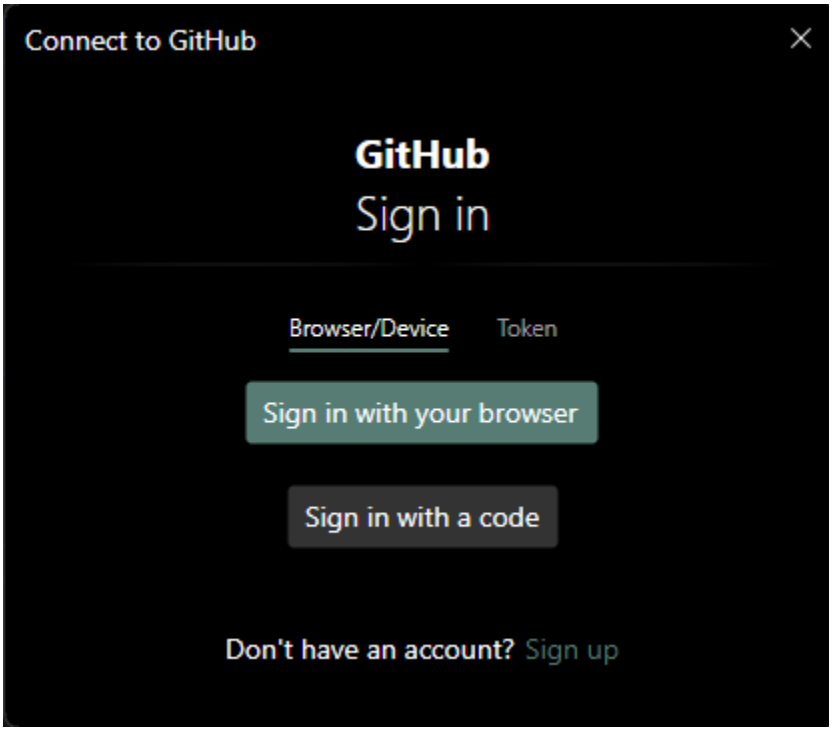
Step 8: click on Open VS Code



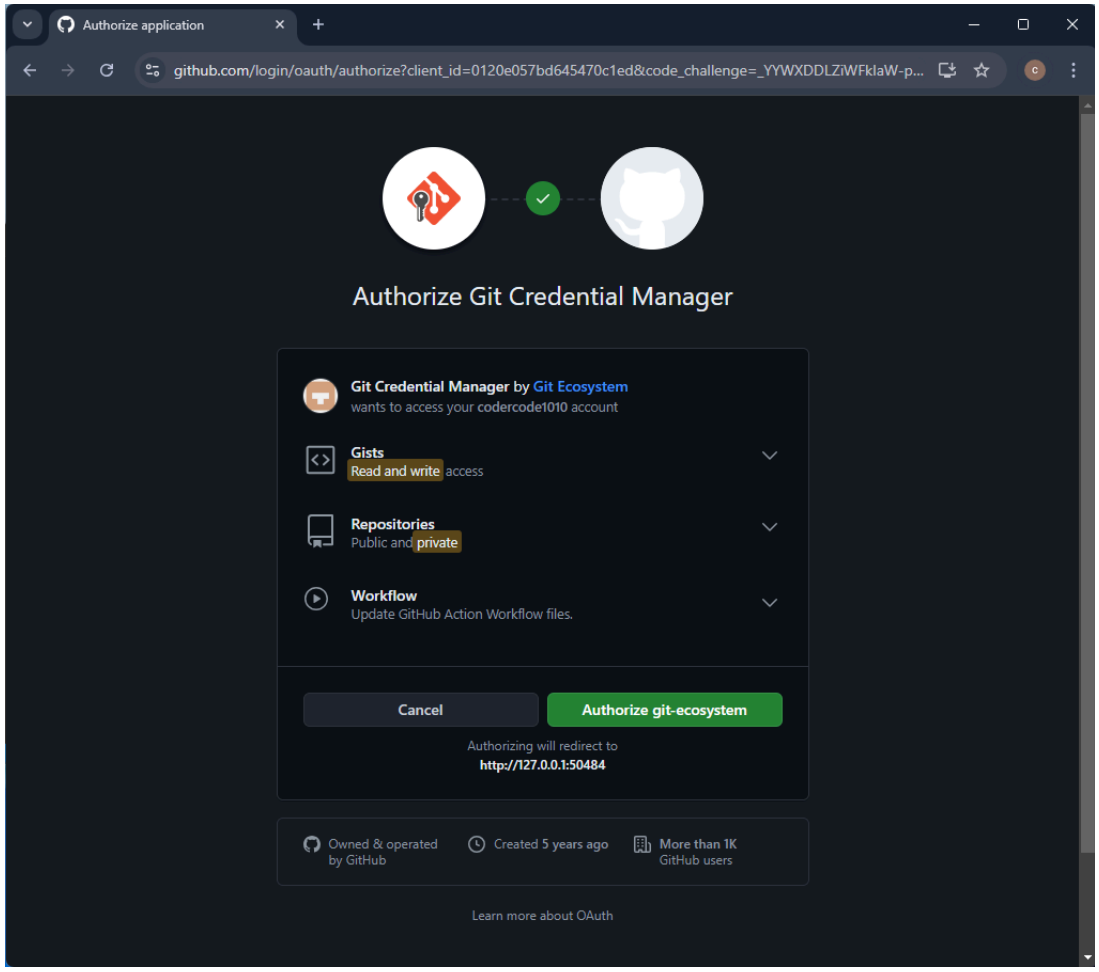
Step 9:Click on Publish Public Repository



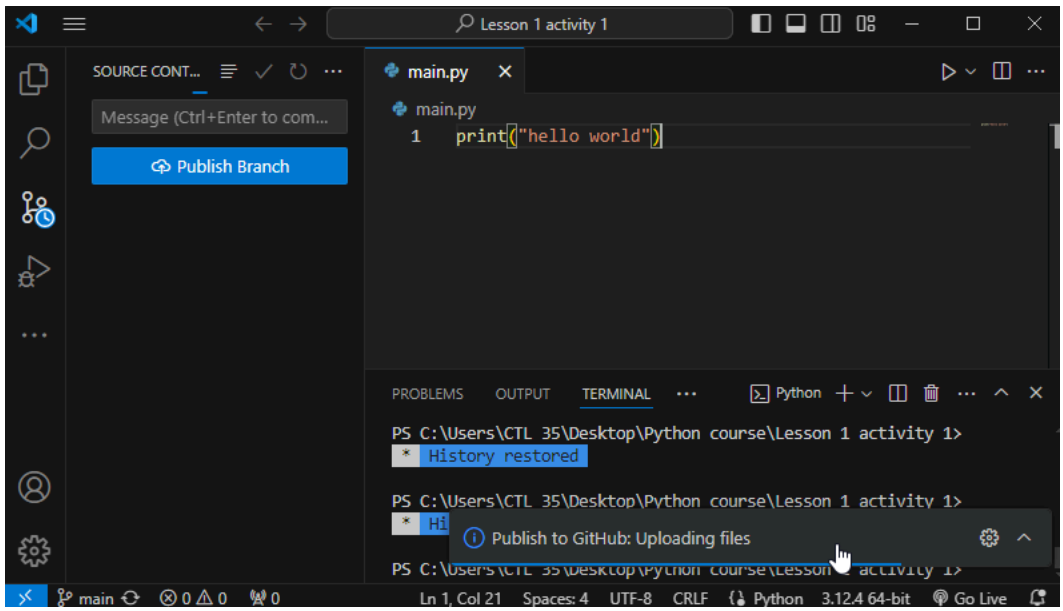
Step 10 : Click on Sign in with your browser



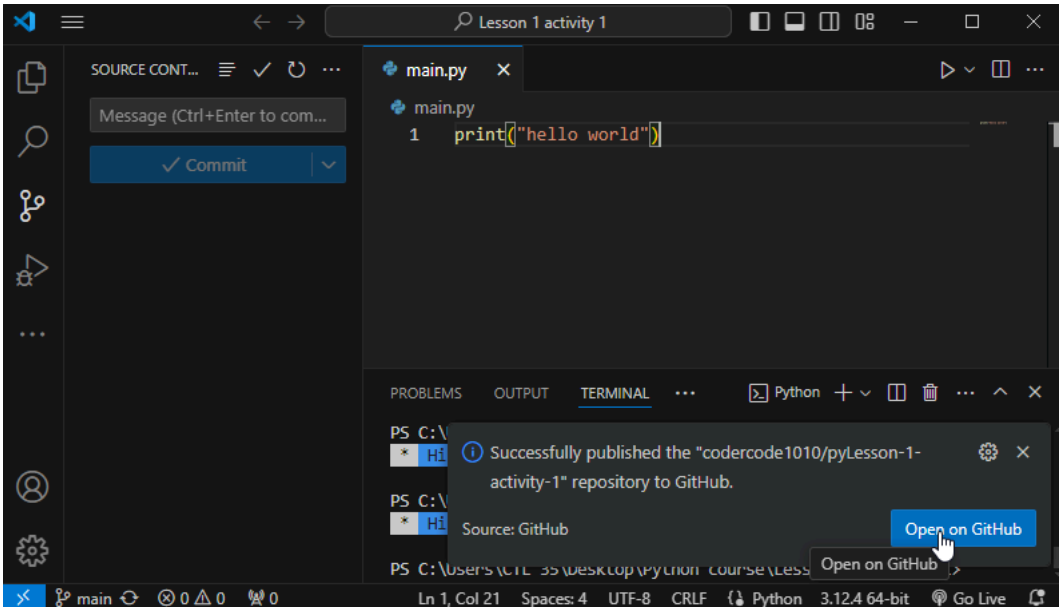
Step 11 : click on Authorize git-ecosystem



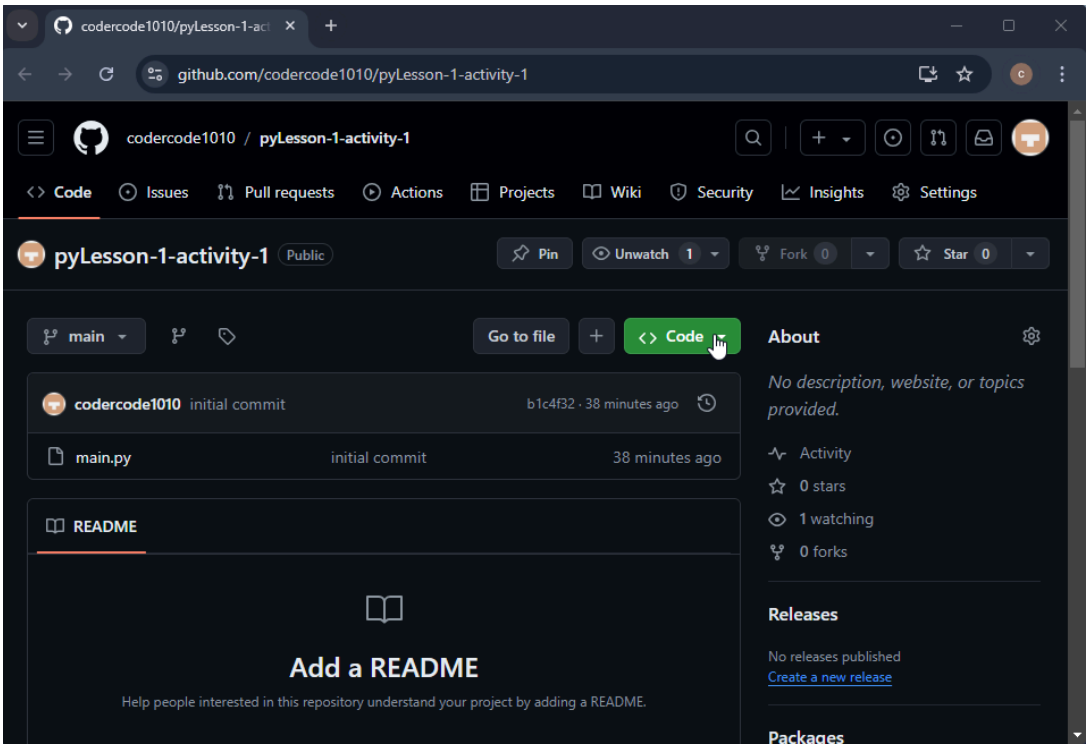
Step 12 : Wait for the files to Upload



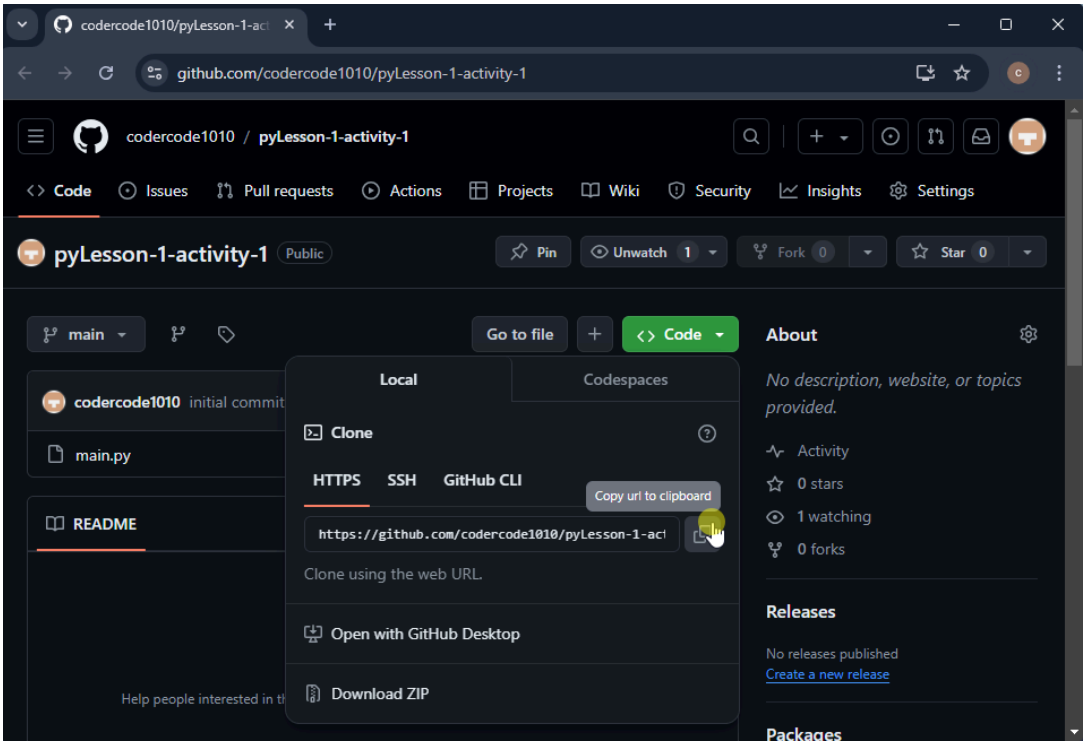
Step 13 : Click on open on GitHub button



Step 14 : Click on the Green Color Code Button



Step 15 : Click on Copy the Code to Clipboard and submit the activity in the classroom



What is an Algorithm, and Why is It Important in AI?

An algorithm is a step-by-step procedure or set of rules designed to solve a specific problem or perform a task. In programming and AI, algorithms are like "recipes" that take input data and follow steps to produce an output or solution.

Importance in AI:

Algorithms are essential in AI as they define the logic behind how AI models make decisions, learn patterns, and improve over time.

They are the backbone of AI techniques like machine learning, which uses algorithms to process and analyze data, identify patterns, and make predictions.

Efficient algorithms ensure AI systems run faster and more accurately, which is critical when working with large datasets.

Here's a simple algorithm to add two numbers:

Start

Input the first number, num1.

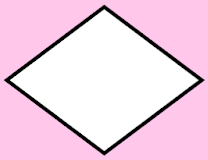
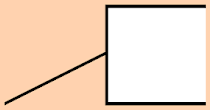
Input the second number, num2.

Process: Add num1 and num2 and store the result in sum.
sum = num1 + num2
Output the result stored in sum.
End

This straightforward algorithm takes two numbers, adds them, and displays the result.

What is a Flowchart, and Why is It Important in AI?

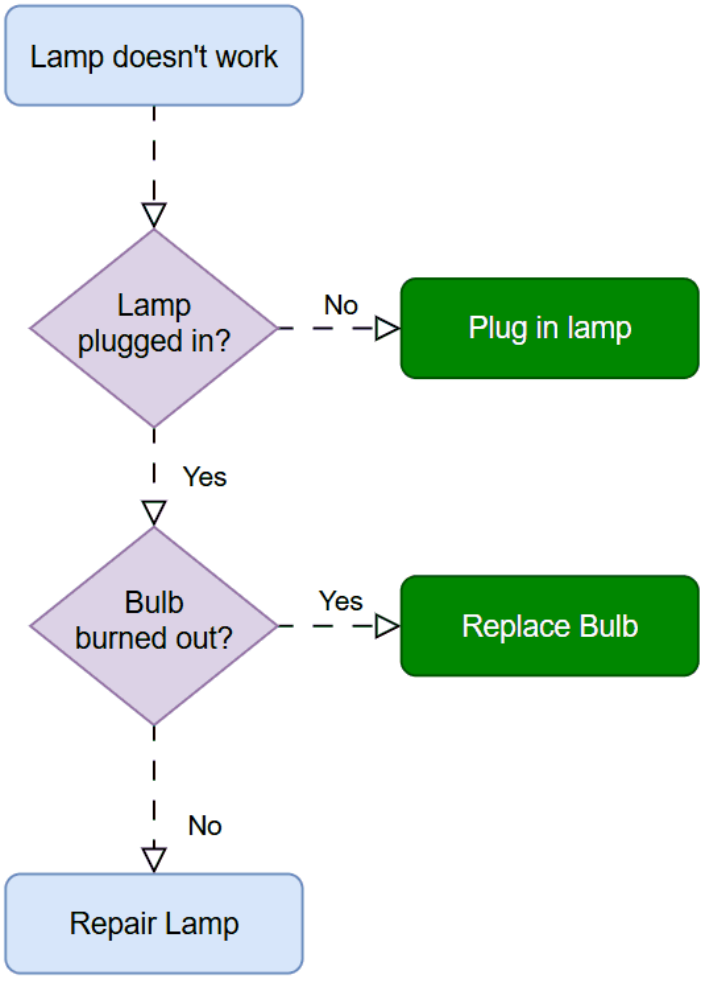
A flowchart is a diagram that visually represents the steps of a process or algorithm using shapes and arrows. It provides a clear, structured way to map out each decision point, action, or step in a process.

ANSI/ISO Shape	Name	Description
	**Flowline (Arrowhead)**	Shows the process's order of operation. A line coming from one symbol and pointing at another. Arrowheads are added if the flow is not the standard top-to-bottom, left-to right.
	**Terminal**	Indicates the beginning and ending of a program or sub-process. Represented as a stadium, oval or rounded (fillet) rectangle. They usually contain the word "Start" or "End", or another phrase signaling the start or end of a process, such as "submit inquiry" or "receive product".
	**Process**	Represents a set of operations that changes value, form, or location of data. Represented as a rectangle
	**Decision**	Shows a conditional operation that determines which one of the two paths the program will take. The operation is commonly a yes/no question or true/false test. Represented as a diamond(rhombus).
	**Input/Output**	Indicates the process of inputting and outputting data, as in entering data or displaying results. Represented as a rhomboid.
	**Annotation (Comment)**	Indicating additional information about a step in the program. Represented as an open rectangle with a dashed or solid line connecting it to the corresponding symbol in the flowchart.
	**Predefined Process**	Shows named process which is defined elsewhere. Represented as a rectangle with double-struck vertical edges.

Importance in AI:

Flowcharts help in designing and understanding complex AI algorithms by breaking them down into individual, understandable steps.
They allow AI developers to visualize the logic flow and decision-making processes, which is crucial for debugging, optimization, and collaboration.
Flowcharts help identify inefficiencies or decision points in AI algorithms, making it easier to optimize or modify the process.

Example :



Note: Use draw.io : Click here to create Flowcharts

Python refresher

Basic Data Types

Python supports standard operators:

Arithmetic operators: +, -, *, /, // (floor division), % (modulus), ** (exponentiation)

Comparison operators: ==, !=, <, >, <=, >=

Logical operators: and, or, not

These data types are fundamental in handling numerical data, text, and labels, which are essential for training AI models and processing inputs.

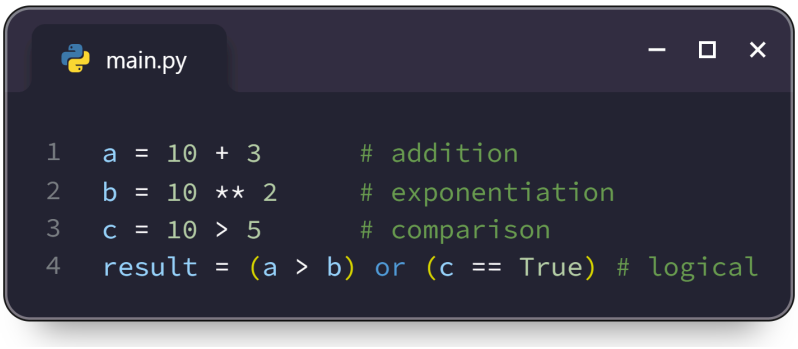
Basic Operators

Python supports standard operators:

Arithmetic operators: +, -, *, /, // (floor division), % (modulus), ** (exponentiation)

Comparison operators: ==, !=, <, >, <=, >=

Logical operators: and, or, not



Operators perform mathematical and logical operations critical for calculations in algorithms, loss functions, and model optimizations in AI.

Conditional Statements

Use if, elif, and else to control flow based on condition

main.py

```python
x = 10
if x > 0:
    print("Positive")
elif x == 0:
    print("Zero")
else:
    print("Negative")
```

Conditional logic helps in decision-making within algorithms, allowing models to branch based on data conditions, such as early stopping in training.

Loops

Python provides for and while loops:

For loop: Used to iterate over a sequence (like a list or range)

While loop: Continues as long as a condition is True

main.py

```python
# For loop
for i in range(5):  # range(5) generates 0,1,2,3,4
    print(i)

# While loop
count = 0
while count < 5:
    print(count)
    count += 1

```

Loops enable iterative processes, which are vital for training models through epochs and processing large datasets in AI.

Functions

Define reusable blocks of code with def:

main.py

```
1  def greet(name):
2      return f"Hello, {name}!"
3
4  print(greet("Alice"))
5
```

Functions allow modular code, making it easier to build, test, and reuse parts of machine learning pipelines (e.g., pre-processing and feature extraction).

Lists

Lists are ordered, mutable collections.

main.py

```
1  my_list = [1, 2, 3, 4, 5]
2  print(my_list[0])  # Access first element
3  my_list.append(6)  # Add element
4  my_list.pop(2)     # Remove element at index 2
```

Lists store data collections (like batches of samples), which can then be processed in bulk for training and inference.

Dictionaries

Dictionaries store key-value pairs.

main.py

```
1  my_dict = {"name": "Alice", "age": 25}
2  print(my_dict["name"])    # Access value by key
3  my_dict["age"] = 26       # Modify value
4  my_dict["city"] = "Paris" # Add new key-value pair
```

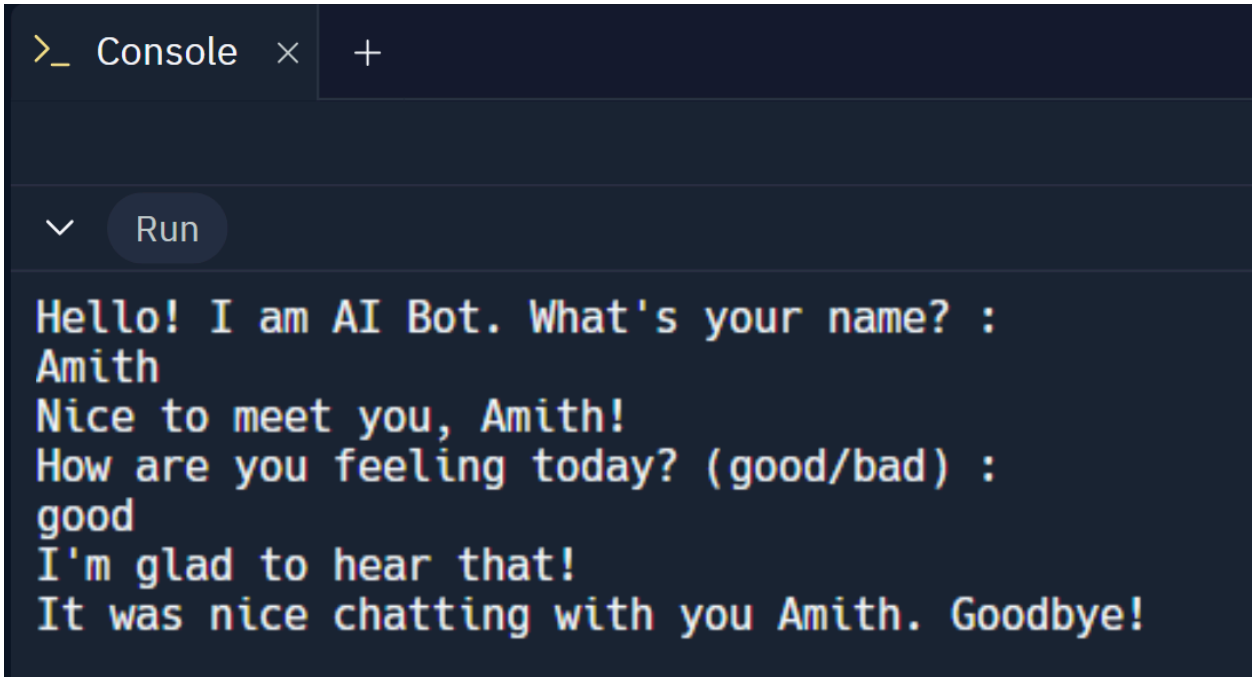
Dictionaries store structured data, often used to label and organize features, hyperparameters, or model configurations in AI projects.

LESSON ACTIVITIES:
Activity 1 : Hello AI
Code : AIEPCM1L1A1
Outline :
Create a basic Python program that simulates an AI chatbot, interacting with users through text and responding intelligently based on their input.
Platform / bpc url :
Video url :
Solution :

main.py

```python
# Greet the user
print("Hello! I am AI Bot. What's your name? : ")

# Get user input
name = input()

# Respond to the user's name
print(f"Nice to meet you, {name}!")

# Ask a question
print("How are you feeling today? (good/bad) : ")
mood = input().lower()

# Use conditional statements to respond based on input
if mood == "good":
    print("I'm glad to hear that!")
elif mood == "bad":
    print("I'm sorry to hear that. Hope things get better soon.")
else:
    print("I see. Sometimes it's hard to put feelings into words.")

# End the conversation
print(f"It was nice chatting with you {name}. Goodbye!")
```

Assets :
OUTPUT:



Activity Explanation :

Greeting the User:

print("Hello! I am AI Bot. What's your name? : ")

This line uses the print() function to display a message on the screen. Here, we introduce the bot and prompt the user for their name. It establishes interaction, a fundamental aspect of AI, making the program more engaging.

Getting User Input:

name = input()

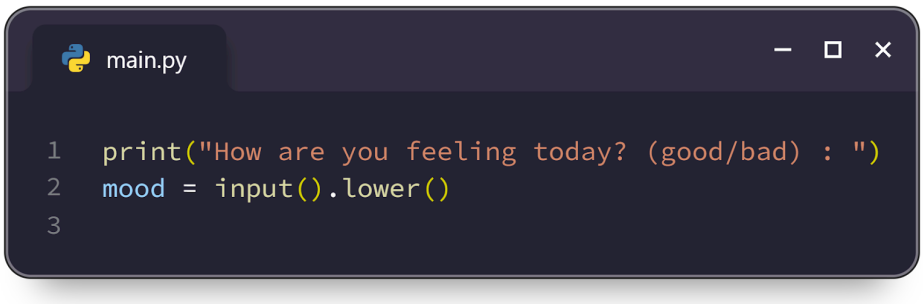
The input() function allows the program to wait for the user to type something, in this case, their name. Storing this input in a variable (name) is crucial as it lets the program remember the user's response, enabling a personalized experience.

Personalized Response:

print(f"Nice to meet you, {name}!")

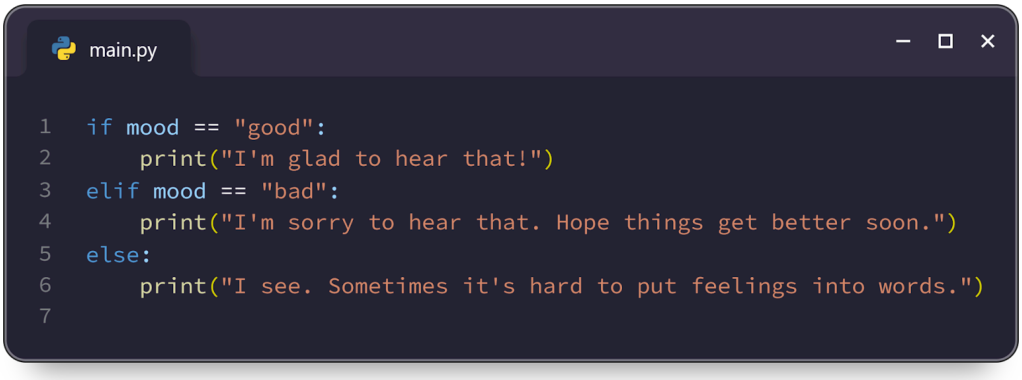
Here, f-string formatting (formatted string literals) is used to incorporate the user's name directly into the response. This simple step gives the illusion that the bot "knows" the user, which enhances user engagement, a fundamental part of AI interaction.

Asking about the User’s Mood:



The bot then asks the user about their mood. The lower() method converts the input to lowercase, ensuring that the user's response is consistent for the upcoming conditions. This demonstrates basic data manipulation, an essential programming skill that ensures the bot can handle varied input reliably.

Conditional Responses (Using if, elif, and else):



Here, the code uses conditional statements (if, elif, and else) to make decisions based on the user’s input. This mimics a basic level of "understanding" where the bot changes its response based on what the user says. These statements are fundamental in programming and AI, allowing computers to make decisions and adjust behavior based on specific criteria.

Ending the Conversation:

print(f"It was nice chatting with you {name}. Goodbye!")

The bot concludes with a friendly farewell, again using the name variable to make it feel personal.

How This Activity Relates to AI

This exercise is an initial step into understanding AI, as it introduces the concepts of input, decision-making, and conditional responses. Though this is a simple script, it reflects the foundation of more complex AI-driven conversations and chatbots used in the

real world. Students learn how AI can interact with users in a personalized way by responding to specific inputs and conditions, which is the basis of creating intelligent assistants and chatbots. By understanding this basic structure, students start grasping how AI can simulate human-like conversations, recognize user inputs, and make decisions based on predefined logic—core concepts that will be expanded upon as they delve deeper into AI.

Learning Outcomes of Hello AI: Introduction to AI & Setting Up Dev Tools

• Gain a foundational understanding of Artificial Intelligence (AI) and its historical development.
• Explore real-world AI applications, understanding AI's impact across various industries.
• Understand programming basics and the purpose of Python as a language in AI.
• Set up essential tools for AI development, including Python, Visual Studio Code, Git, and GitHub.
• Establish a well-organized folder structure for efficient code management.
• Learn Git and GitHub basics to track changes, collaborate, and push code to repositories.
• Master Python programming fundamentals, covering data types, operators, conditionals, loops, functions, lists, and dictionaries.
• Develop algorithms and flowcharts to map and visualize code logic.
• Create a simple Python-based AI chatbot for hands-on experience with AI programming basics.

Teacher Agenda

Important Links
Activity 1 Hello AI: Click here

Let's Build Better Lessons Together

Hey Teachers,
Your insights matter! Please take a moment to share your thoughts on this lesson. The more specific you are, the faster we can improve both student learning and your teaching experience.

Submit feedback now!

Thank you!
– Curriculum Team

Highlights:

- Always appreciate and motivate the students.
- Keep the sessions engaging and interactive.
- Explain the concepts clearly and guide students through activities.

Agenda and Topics Covered:

- History of AI
- Setting up Python
  - Download Python: Link
  - Installation steps with screenshots:
    - Ensure checkboxes are marked and click "Install Now."
    - Wait for installation to complete and click "Close."
- Setting up VS Code
  - What is VS Code:
    - A free code editor created by Microsoft.

- Features: Text Editing, Extensions, Debugging, Customization.

- Download and Install VS Code: Link

- Installation steps with screenshots.

- Basic settings: Theme, Auto Save, Word Wrap, Format on Paste & Save.

Setting up Git & GitHub

- Introduction to Git:

  - A tool to track changes to your code.

  - Benefits: Save Snapshots, Track Changes, Revert Changes.

- Download and Install Git: Windows | Mac

- Installation steps with screenshots.

- Introduction to GitHub:

  - A platform to store and share Git projects online.

  - Benefits: Collaborate, Share, Contribute.

- Creating a GitHub Account:

  - Step-by-step guide with screenshots.

Folder Structure

- Importance:
  - Organisation, Clarity, Efficiency.
- Creating and organising folders in VS Code:
  - Step-by-step guide to creating folders and files.

Install & Use Extensions in VS Code

- Install Python Extension.
- Run Python.
- Install Live Share for collaboration.
- Install Live Preview & Live Server to view the output

Git Configuration

- Configuring username and email in Git:
  - Use git config commands in VS Code terminal.

Push Code to GitHub

- Initialising a repository:
  - Tell Git to watch the folder.

Staging and committing changes.

Publishing the repository to GitHub:

Step-by-step guide with screenshots.

Additional Notes:

Explain the concepts clearly while guiding students through activities.

Summarise the concepts learned at the end of the class.

Lesson Quiz

Q1) What was the purpose of the Dartmouth Conference in 1956?
Correct answer:
To officially establish Artificial Intelligence as a field of study.
Explanation:
The Dartmouth Conference in 1956 marked the birth of AI, bringing researchers together to discuss and define the concept of machine intelligence.
Options:
To create the first AI robot
To establish AI as a field of study
To invent the Turing Test
To start the AI Winter
Q2) Who proposed the first model of artificial neurons in 1943?
Correct answer:
McCulloch and Pitts
Explanation:
McCulloch and Pitts proposed a model for artificial neurons, laying foundational work for artificial neural networks.
Options:
Alan Turing
McCulloch and Pitts
John McCarthy
Marvin Minsky

Q3) What was a significant development in AI during the 1990s?
Correct answer:
IBM’s Deep Blue defeated world chess champion Garry Kasparov.
Explanation:
IBM’s Deep Blue’s victory over Garry Kasparov in 1997 was a landmark achievement, demonstrating AI’s potential in handling complex strategic tasks.
Options:
The invention of the Turing Test
Creation of the first AI robot
Deep Blue defeated Garry Kasparov
Introduction of Siri
Q4) What is “AI Winter”?
Correct answer:
A period when AI research slowed due to reduced funding and interest.
Explanation:
"AI Winter" refers to periods during the 1970s-1980s when high expectations were not met, leading to funding cuts and slower progress in AI.
Options:
A phase of rapid AI development
A period of reduced funding and progress in AI
The season when AI conferences are held
The launch of virtual assistants
Q5) What role does deep learning play in AI?
Correct answer:
It enables AI to recognize images, understand language, and perform complex tasks by mimicking the human brain.
Explanation:
Deep learning, a subset of machine learning, uses neural networks to tackle tasks like image recognition, language translation, and more by simulating human brain processes.
Options:
Helps AI identify colors
Allows AI to mimic human brain functions
Used for basic arithmetic in AI
Helps create web applications
VS Code, Git & GitHub Questions
Q6) What is the primary function of Git in software development?
Correct answer:
To track changes in code, enabling version control.
Explanation:

Git allows developers to track code changes, revert to previous versions, and collaborate effectively by saving "snapshots" of code.
Options:
To speed up code execution
To edit code in real-time
To track changes in code
To compile programs
Q7) What feature in Visual Studio Code allows developers to add extra functionality, such as debugging and live previews?
Correct answer:
Extensions
Explanation:
VS Code has an Extensions feature that allows developers to add functionalities like live previews, code sharing, and more to enhance the coding experience.
Options:
Sidebar
Extensions
Menu Bar
Command Line
Q8) Why are flowcharts important in AI development?
Correct answer:
They help visualize complex AI algorithms and processes, making them easier to understand and debug.
Explanation:
Flowcharts are useful for visualizing the steps and decision points in AI algorithms, which aids in understanding, debugging, and optimizing the AI workflow.
Options:
They add colors to code
They organize files in a project
They help visualize AI algorithms
They control the speed of AI programs
Q9) Which of the following is a valid variable name in Python?
Correct answer:
my_variable
Explanation:
Python variable names can contain letters, numbers, and underscores but must not start with a number or contain spaces.
Options:
my variable
2variable

my_variable
Variable-name
Q10) What is the correct output of the following code?
python
Copy code
x = 7
y = 3
print(x * y)
Correct answer:
21
Explanation:
The code multiplies x and y, which are assigned values 7 and 3, respectively, resulting in 21.
Options:
10
21
24
30

After-Class Project: Enhanced AI Chatbot

Goal

Extend the basic chatbot by adding more interactive features, including emotion-based responses, additional conversation topics, and an option to repeat the conversation or end it. This project will reinforce the use of conditional statements, user input handling, loops, and basic string manipulation.

Getting Started

Open draw.io and create a flowchart outlining the chatbot program logic.

The flowchart should include steps for greeting the user, handling mood-based responses, asking follow-up questions, and providing an option to continue or exit.

Once the flowchart is complete, open Replit or your preferred IDE and create a new project.

Name your project (e.g., **Enhanced Chatbot**).

Implement the activity as a Python file within this project.

Instructions

Activity: Enhanced Chatbot

Create a flowchart in draw.io:

Start with greeting the user and asking for their name.

Add branches to handle responses based on mood (good, bad, or neutral).

Include follow-up questions, such as asking about hobbies or favorite activities.

Add a loop for the option to continue chatting or exit the conversation.

End the flowchart with a personalized farewell message.

After completing the flowchart, implement the chatbot in Python:

Use conditional statements and loops based on the flowchart logic.

Hints

Use loops (e.g., `while` loop) to keep the conversation going until the user chooses to exit.

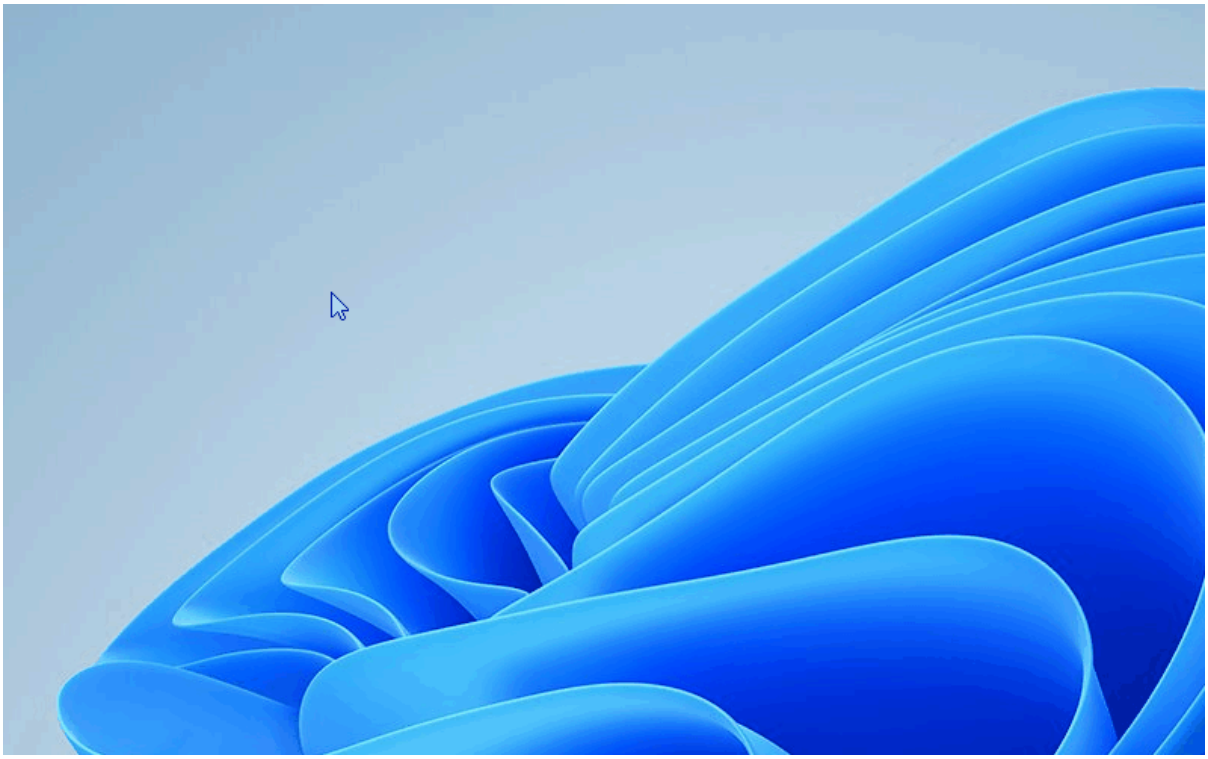
Use `input().lower()` to handle case sensitivity for user input.

Consider using `elif` statements to add more branching based on user responses.

Solution : Click here

HOW TO SUBMIT ACP

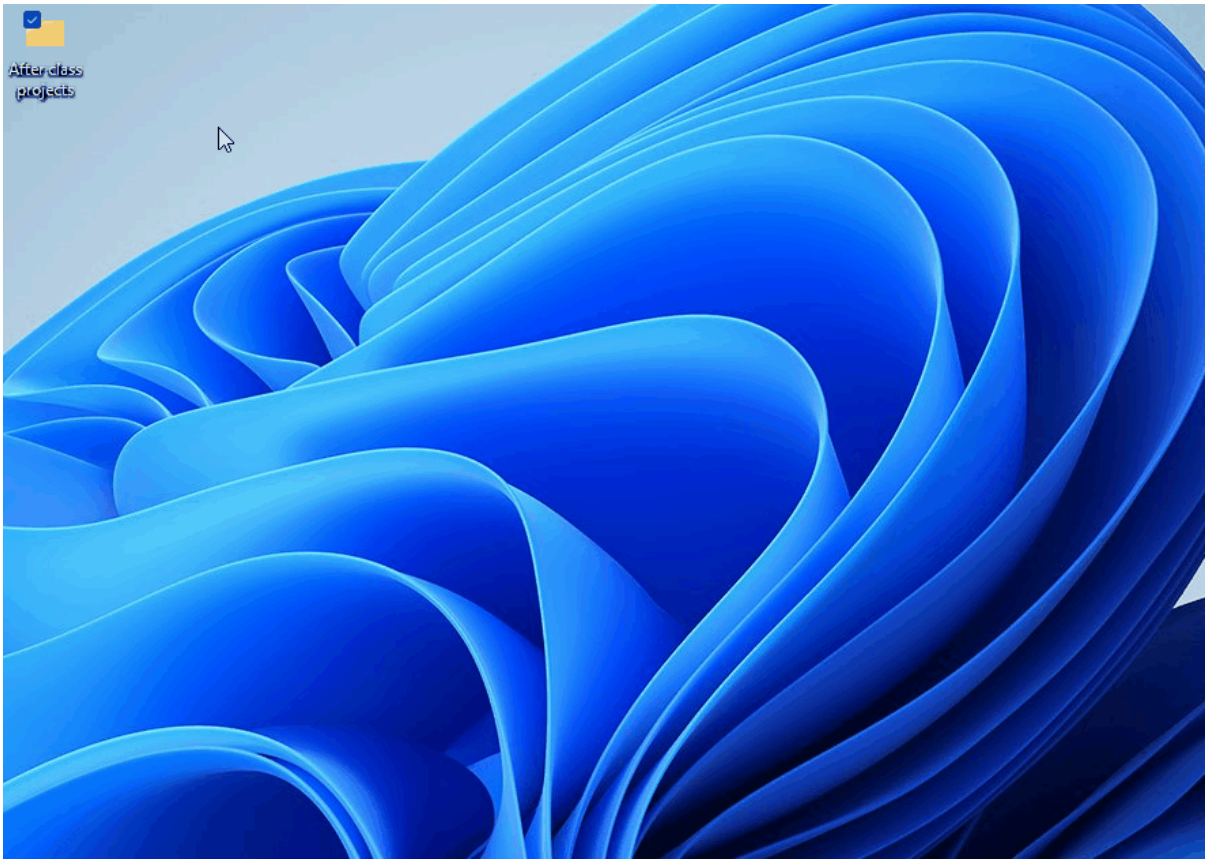
Right Click> Click On New > Click On Folder > Name the folder as After Class Project.



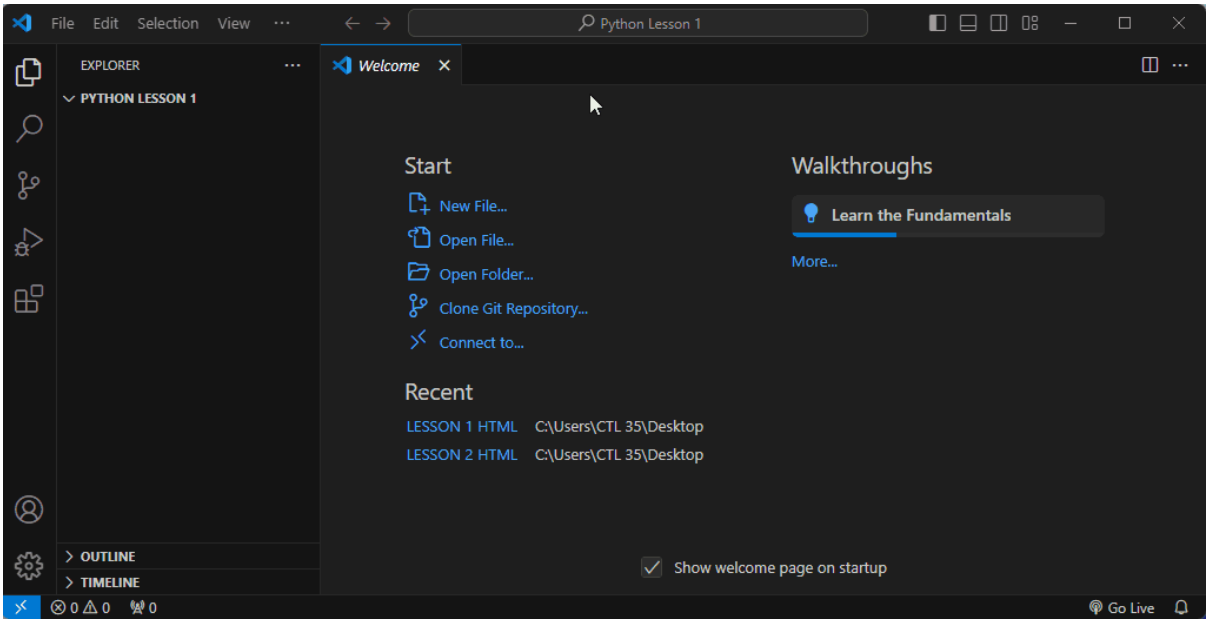
Open the Folder > Right Click > Click on New > Click on Folder > Name the folder with ACP title > Right Click on the Folder > Click on Show more options > Click on Open with Code.

Note: For better understanding you can name the folder exactly as

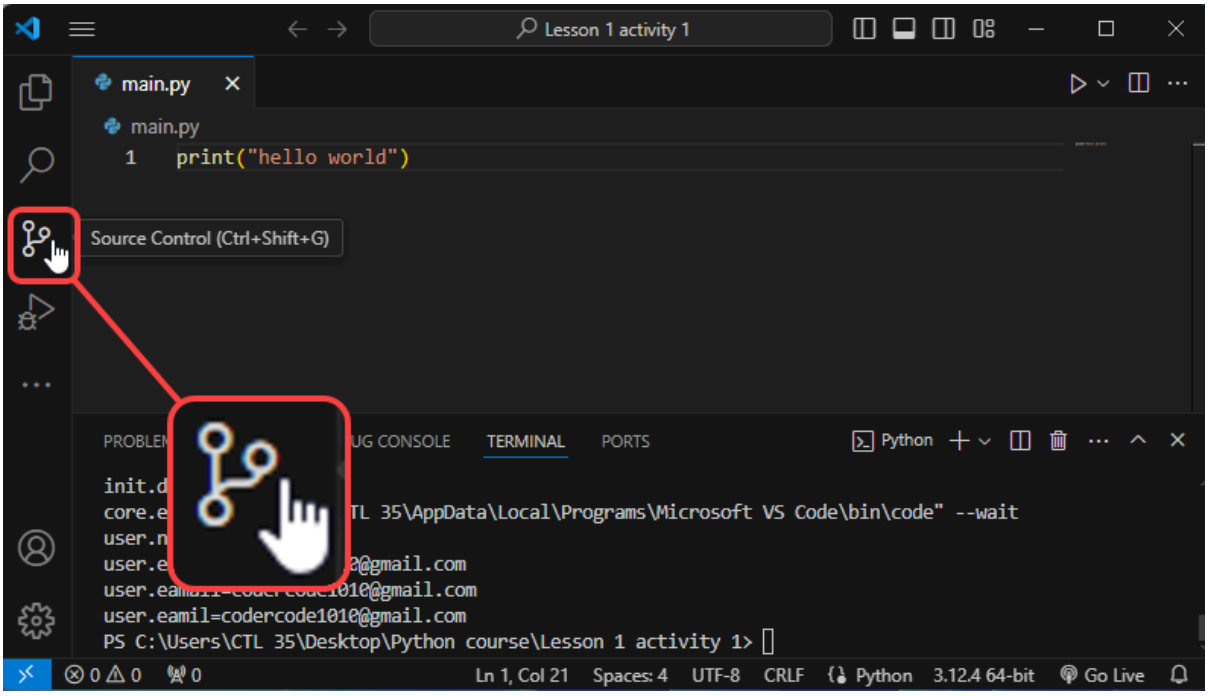
the Project name given in your dashboard



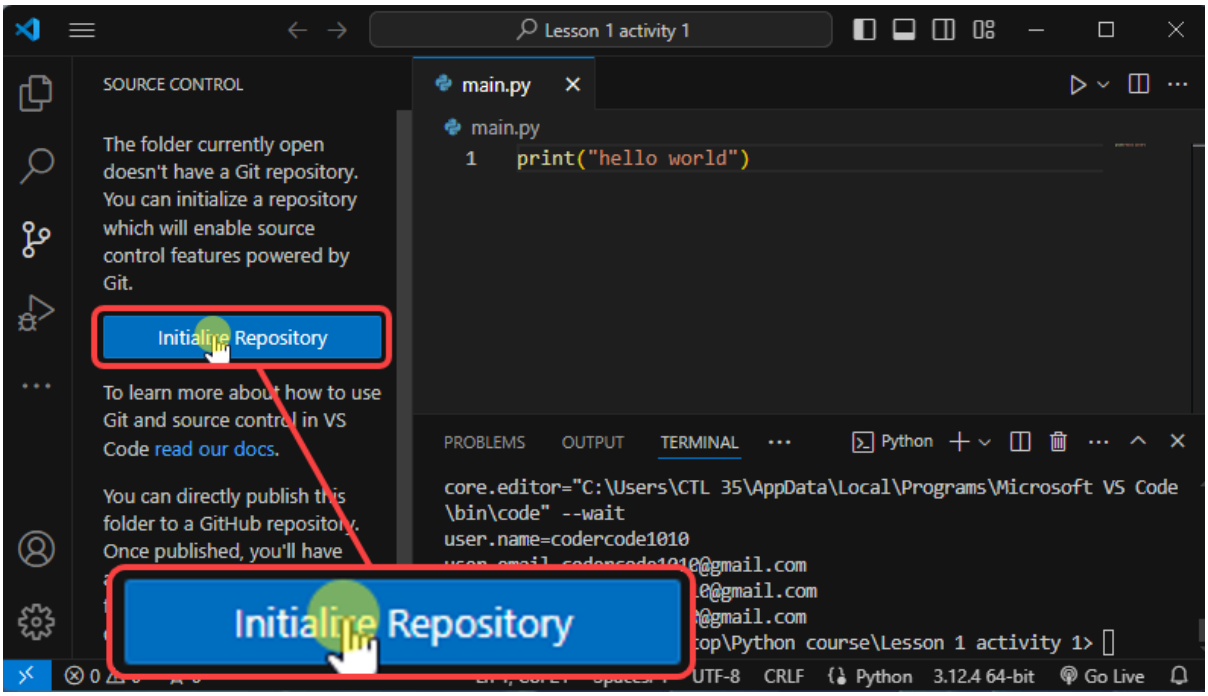
Create the file for the ACP



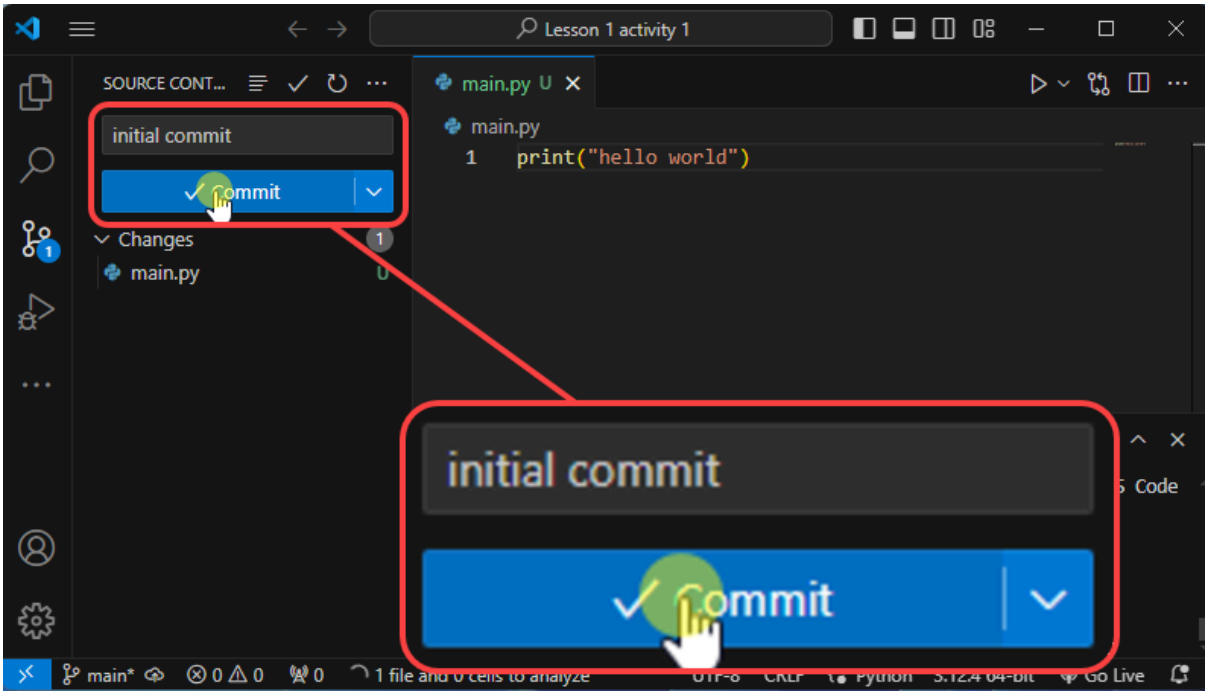
Once you have completed the After Class Project> Navigate to the VS Code Sidebar > Click on Source Control



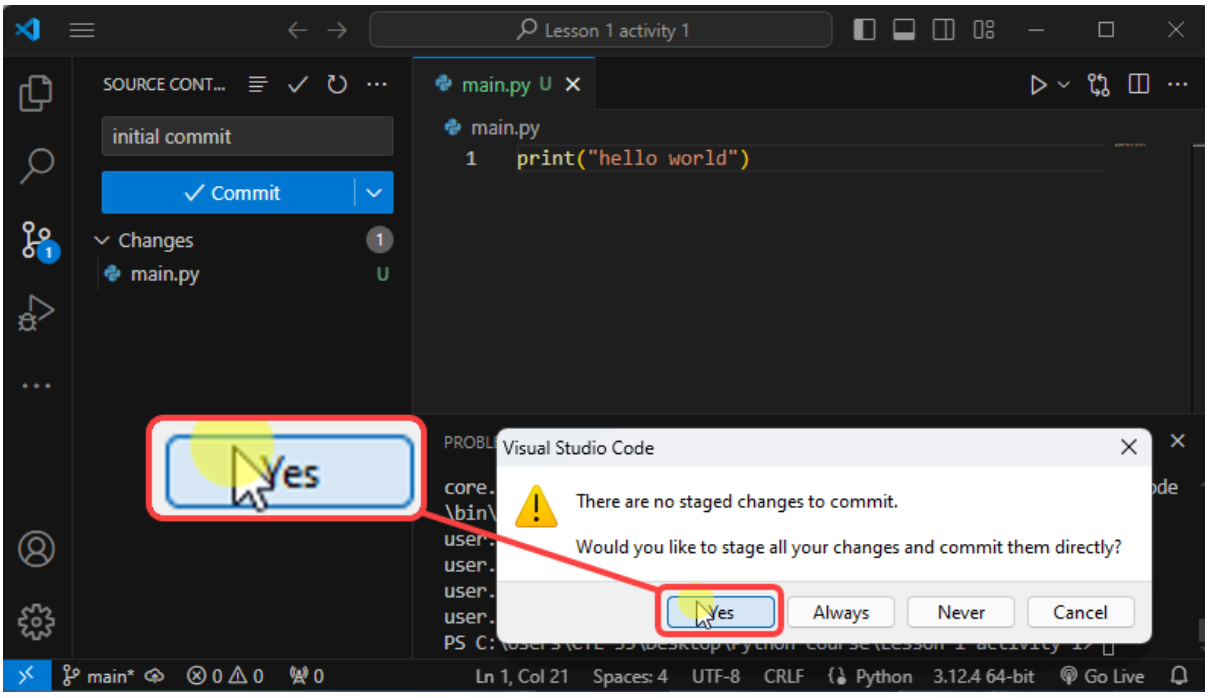
Click on Initialize Repository



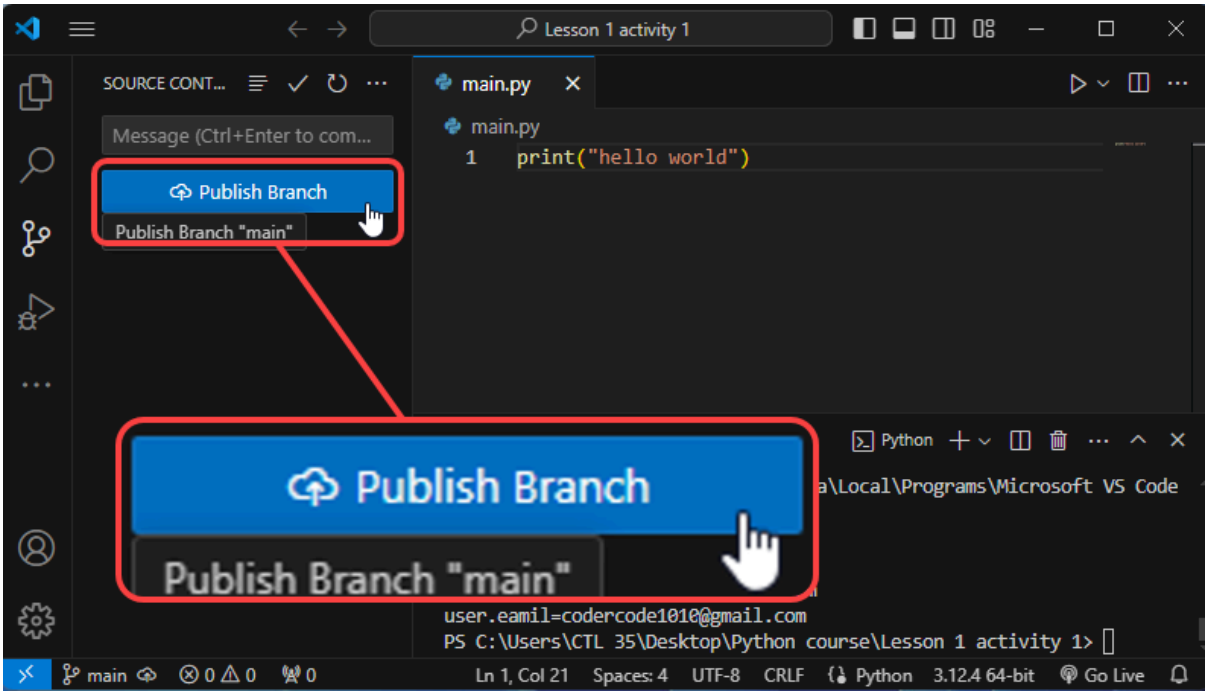
Write the Commit message & Click on commit



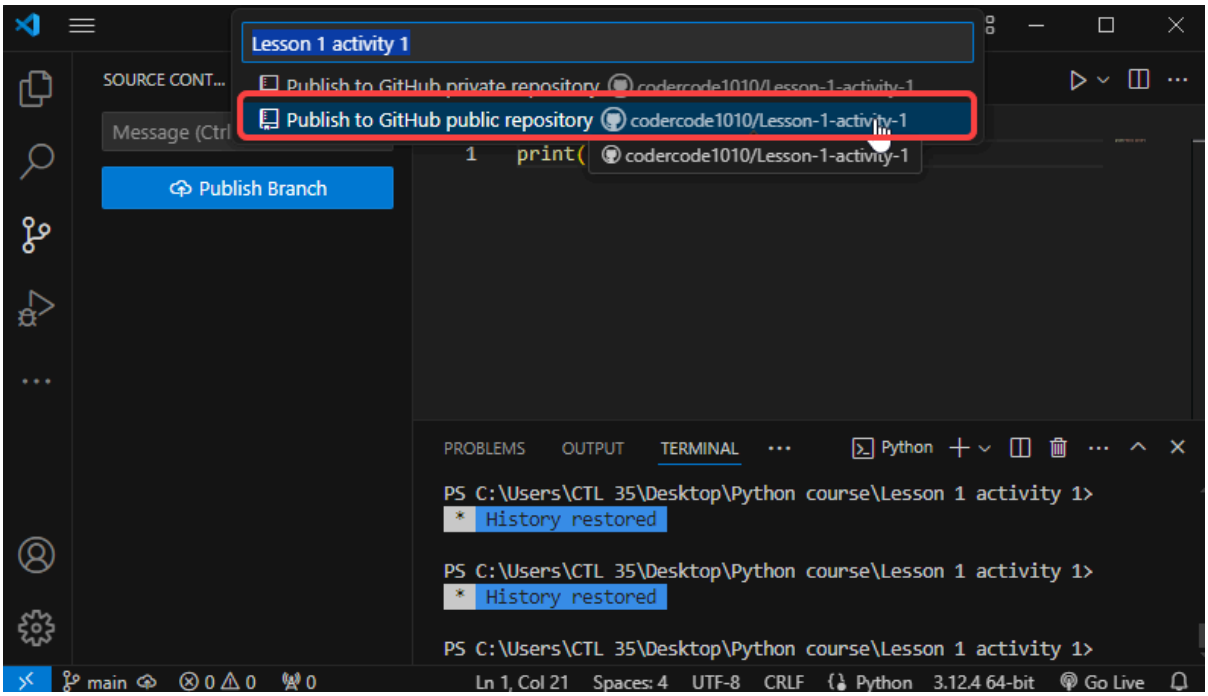
Click on yes to Stage all changes



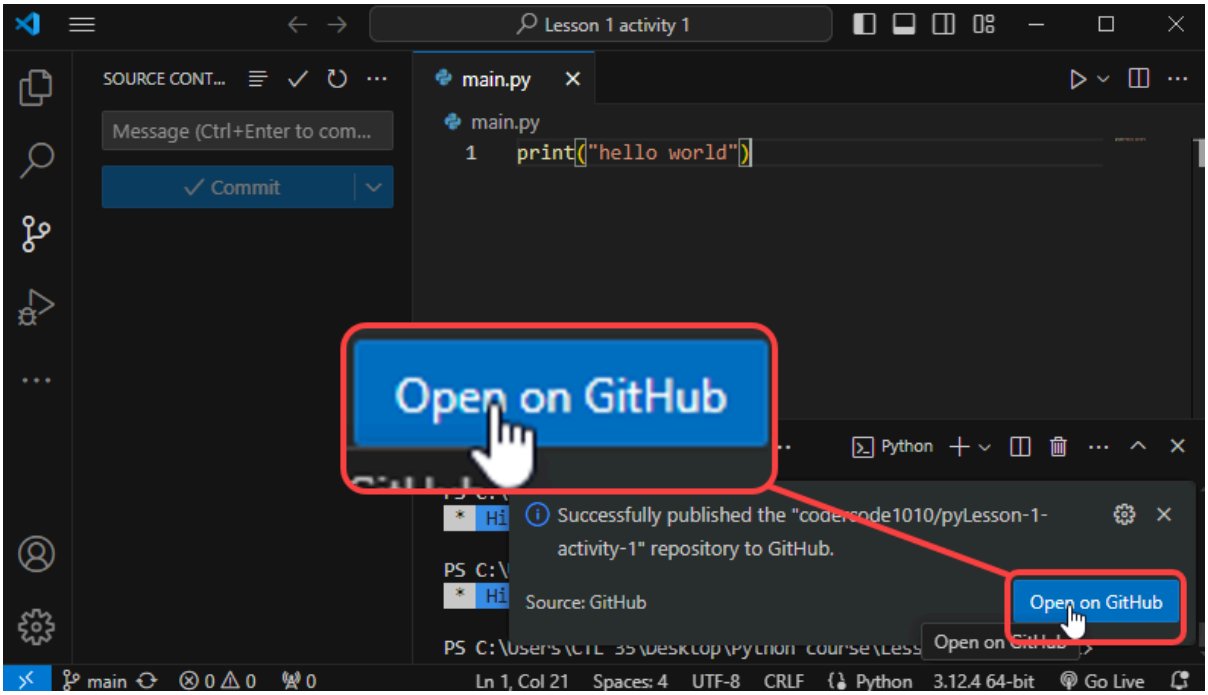
Click on Publish



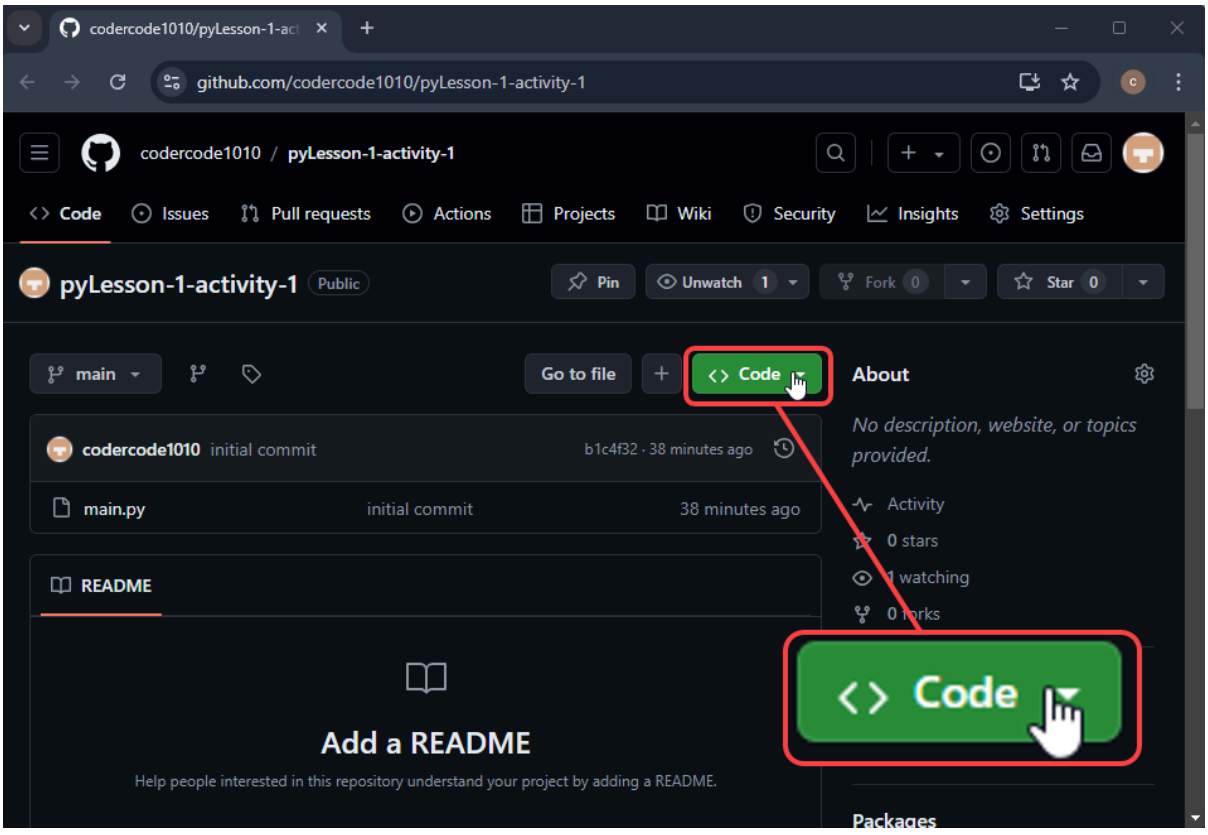
Click on Publish to GitHub public repository & wait for the files to upload



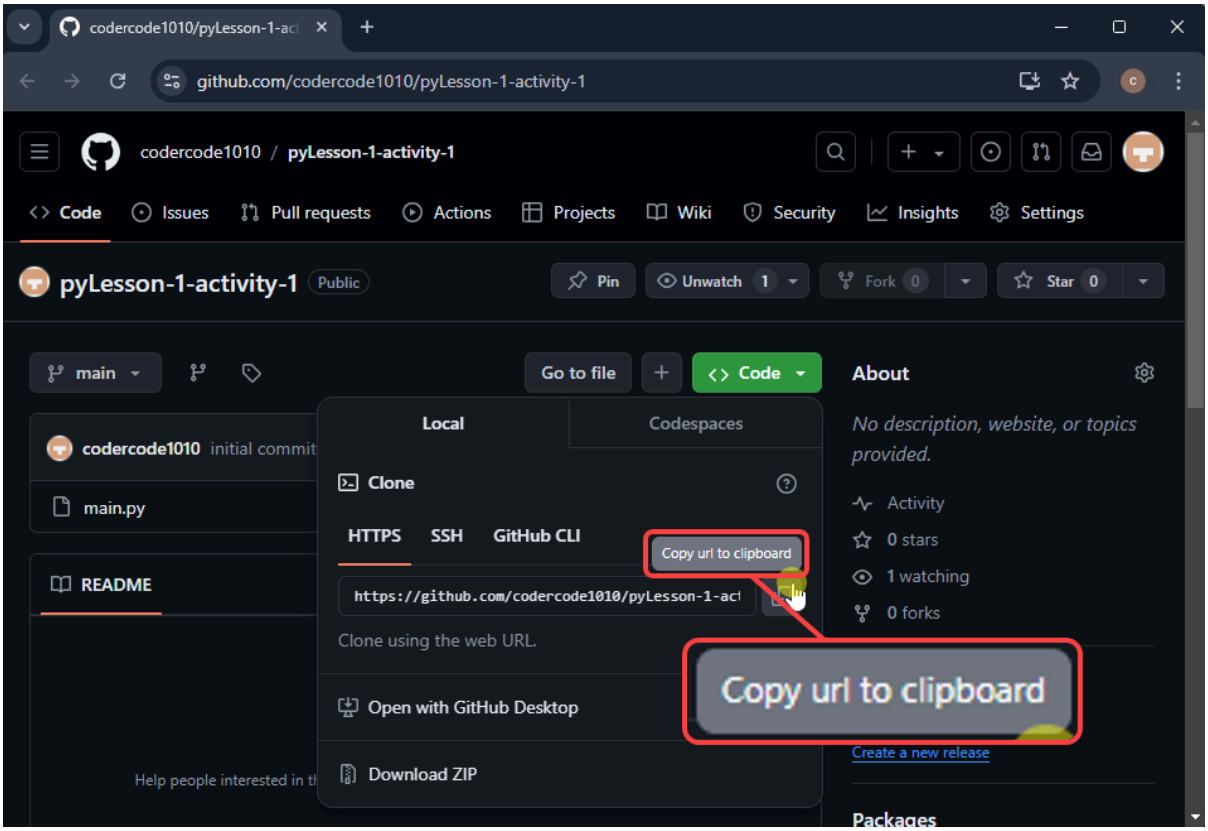
Click on Open on GitHub



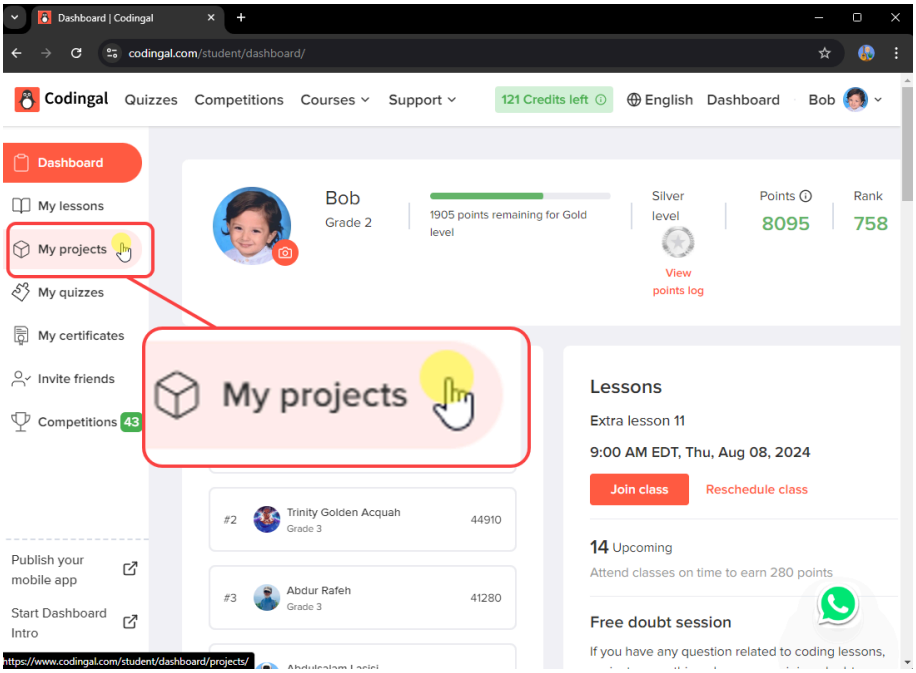
Click on code button



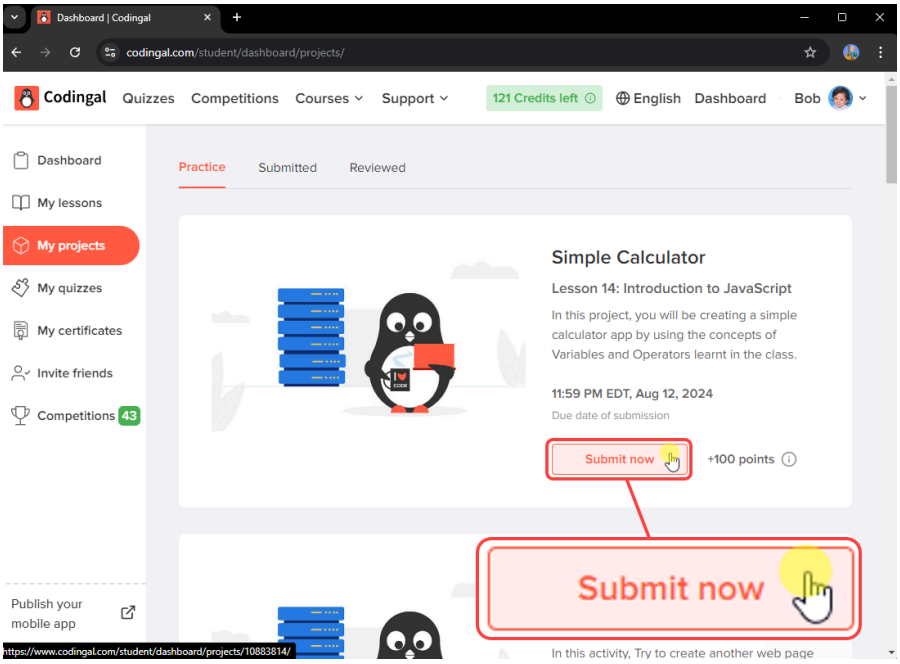
Click on Copy to clipboard to copy the link



Open your dashboard > Click on My projects

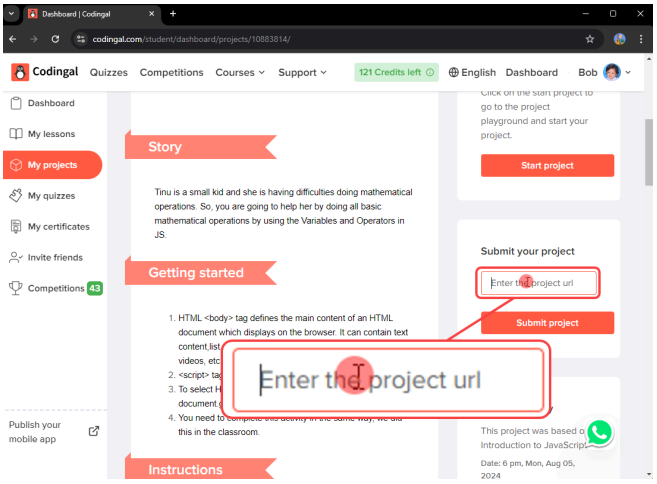


Click on Submit now button

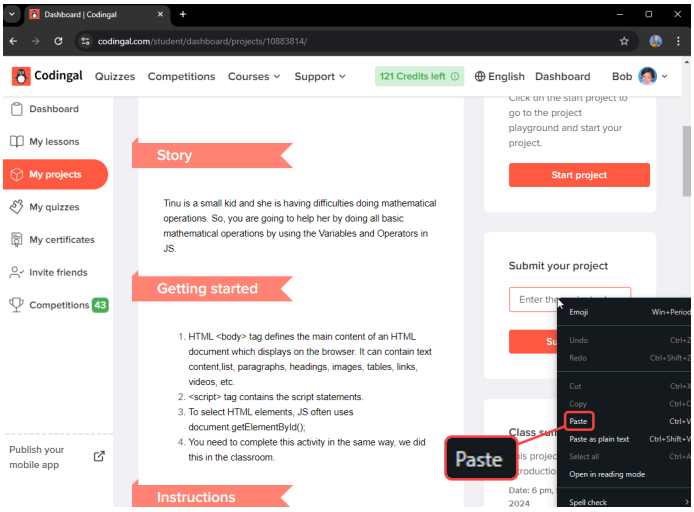


Scroll down > navigate your mouse to the right side of the screen

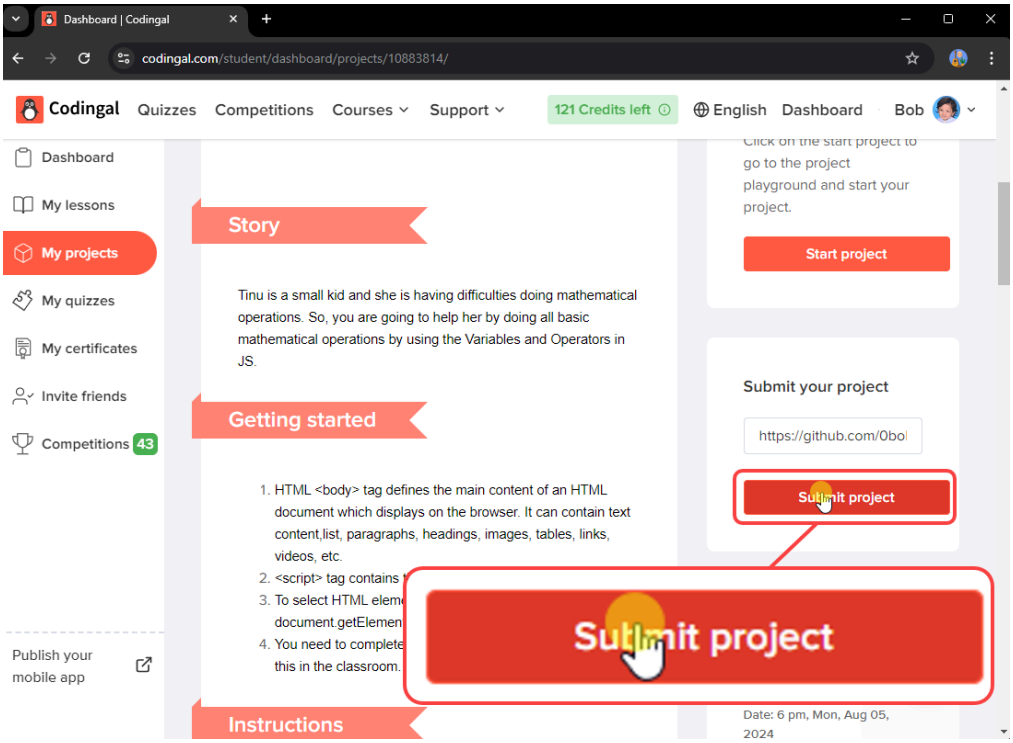
right click on enter the project url



Click on the paste to paste the link



Click on Submit project button to submit the project



Tab 2

Below is a **line-by-line, in-depth** explanation of the final Sentiment Spy code that uses **Colorama** (for color), **Emojis** (for fun icons), and **TextBlob** (for sentiment analysis). Each **library**, **attribute**, **parameter**, **method**, **logic**, and **flow** is explained so you can understand precisely **how** and **why** it all works.

Complete Code

python
CopyEdit

```
import colorama
from colorama import Fore, Style
from textblob import TextBlob
# 1) Initialize colorama for colored output
colorama.init()
# 2) Print an initial greeting with emojis and color
print(f"{Fore.CYAN}👋🎉 Welcome to Sentiment Spy! 🕵️{Style.RESET_ALL}")
# 3) Prompt the user for their name
user_name = input(f"{Fore.MAGENTA}Please enter your name: {Style.RESET_ALL} ").strip()
if not user_name:
    user_name = "Mystery Agent"  # If the user didn't type anything, use a default name
# 4) A list to store all conversation entries (text, polarity, sentiment_type)
conversation_history = []
# 5) Print instructions and an initial greeting to the user
print(f"\n{Fore.CYAN}Hello, Agent {user_name}!")
print(f"I will analyze your sentences with TextBlob and show you the sentiment. 🔍")
print(f"Type {Fore.YELLOW}'reset'{Fore.CYAN}, {Fore.YELLOW}'history'{Fore.CYAN}, "
      f"or {Fore.YELLOW}'exit'{Fore.CYAN} to quit.{Style.RESET_ALL}\n")
# 6) Main loop for user input
while True:
    user_input = input(f"{Fore.GREEN}>> {Style.RESET_ALL}").strip()

    # 6.1) If the user just pressed enter (empty input), prompt again
    if not user_input:
        print(f"{Fore.RED}Please enter some text or a valid command.{Style.RESET_ALL}")
        continue
    # 6.2) Exit command
    if user_input.lower() == "exit":
        print(f"\n{Fore.BLUE}🚪 Exiting Sentiment Spy. Farewell, Agent {user_name}!
🏁{Style.RESET_ALL}")
        break
```

```
# 6.3) Reset command
elif user_input.lower() == "reset":
    conversation_history.clear()
    print(f"{Fore.CYAN}🎉 All conversation history cleared!{Style.RESET_ALL}")
    continue

# 6.4) History command
elif user_input.lower() == "history":
    if not conversation_history:
        print(f"{Fore.YELLOW}No conversation history yet.{Style.RESET_ALL}")
    else:
        print(f"{Fore.CYAN}📜 Conversation History:{Style.RESET_ALL}")
        # Loop through all saved messages
        for idx, (text, polarity, sentiment_type) in enumerate(conversation_history, start=1):
            # Assign color & emoji based on sentiment type
            if sentiment_type == "Positive":
                color = Fore.GREEN
                emoji = "😊"
            elif sentiment_type == "Negative":
                color = Fore.RED
                emoji = "😢"
            else:
                color = Fore.YELLOW
                emoji = "😐"
            print(f"{idx}. {color}{emoji} {text} "
                  f"(Polarity: {polarity:.2f}, {sentiment_type}){Style.RESET_ALL}")
    continue
# 6.5) Any other input → Perform sentiment analysis
polarity = TextBlob(user_input).sentiment.polarity
if polarity > 0.25:
    sentiment_type = "Positive"
    color = Fore.GREEN
    emoji = "😊"
elif polarity < -0.25:
    sentiment_type = "Negative"
    color = Fore.RED
    emoji = "😢"
else:
    sentiment_type = "Neutral"
    color = Fore.YELLOW
    emoji = "😐"
# Add the data to conversation history
conversation_history.append((user_input, polarity, sentiment_type))
```

```
# Print the result of the sentiment analysis to the user
print(f"{color}{emoji} {sentiment_type} sentiment detected! "
      f"(Polarity: {polarity:.2f}){Style.RESET_ALL}")
```

Detailed Explanations

Below is a breakdown of **every** piece—libraries, objects, logic, and flow.

1. Imports and Libraries

```
import colorama
from colorama import Fore, Style
from textblob import TextBlob
```

colorama: A Python library that allows you to print colored text in terminals across different operating systems (Windows, macOS, Linux).

colorama.init() is crucial on Windows to convert ANSI escape sequences into calls to the Windows API, making colors possible on the command prompt.

from colorama import Fore, Style:

Fore: Contains constants like Fore.RED, Fore.GREEN, Fore.YELLOW, etc., which switch the text color in the terminal.

Style: Contains constants like Style.RESET_ALL, which resets styling to defaults.

from textblob import TextBlob:

TextBlob is a simple library for common **natural language processing** tasks, such as **sentiment analysis**, **part-of-speech tagging**, **spelling correction**, etc.

We only use **sentiment** here, specifically the polarity attribute.

2. colorama.init()

```
colorama.init()
```

This call initializes Colorama so that all subsequent print statements that use Fore or Style will correctly display colored text in the terminal, regardless of OS.

3. Initial Greeting

```
print(f"{Fore.CYAN}👋🎉 Welcome to Sentiment Spy! 🕵️{Style.RESET_ALL}")
```

print(...): Outputs text to the console.

f"{Fore.CYAN}...{Style.RESET_ALL}": This is a Python **formatted string literal** (f-string). We insert **ANSI color codes** from Fore.CYAN to make the text cyan, and then we append Style.RESET_ALL at the end to reset the color styling back to normal so we don't affect subsequent prints.

Emojis (👋🎉 and 🕵️) are just Unicode characters that print in most modern terminals.

4. Getting the User's Name

```
user_name = input(f"{Fore.MAGENTA}Please enter your name: {Style.RESET_ALL} ").strip()
if not user_name:
    user_name = "Mystery Agent"
```

input(...): Waits for the user to type something and press Enter, returning the typed string.

Inside the input(...) prompt, we again use an f-string with color:

Fore.MAGENTA changes the prompt to magenta text.

Style.RESET_ALL resets back after printing the prompt.

.strip(): Removes leading and trailing whitespace (like spaces, tabs, or newlines).

if not user_name:: Checks if the string is empty (if the user just pressed Enter). If it is, we default to **"Mystery Agent"**.

5. Conversation History

```
conversation_history = []
```

We keep a **list** named conversation_history. It will store **tuples** of the form **(text, polarity, sentiment_type)**.

Initially, it's empty.

6. Printing Instructions

```
print(f"\n{Fore.CYAN}Hello, Agent {user_name}!")
print(f"I will analyze your sentences with TextBlob and show you the sentiment. 🔍")
print(f"Type {Fore.YELLOW}'reset'{Fore.CYAN}, {Fore.YELLOW}'history'{Fore.CYAN}, "
      f"or {Fore.YELLOW}'exit'{Fore.CYAN} to quit.{Style.RESET_ALL}\n")
```

{Fore.CYAN} and **{Fore.YELLOW}**: We use these to highlight specific words or commands, making them stand out in color.

\n: Adds extra lines for readability.

Emojis like 🔍 are again just Unicode characters placed in the string.

7. Main Interaction Loop

```
while True:
    user_input = input(f"{Fore.GREEN}>> {Style.RESET_ALL}").strip()
    ...
```

We start an **infinite loop** (while True) to continually prompt the user for input.

We color the prompt ">>" in **green** to show that the user can type there. We reset style after the prompt so the user's typed text is in the default color.

.strip() again removes leading and trailing whitespace.

7.1 Checking for Empty Input

```
if not user_input:
    print(f"{Fore.RED}Please enter some text or a valid command.{Style.RESET_ALL}")
    continue
```

If user_input is an empty string (meaning the user pressed Enter without typing anything), we show a warning and use continue to jump back to the top of the loop for the next prompt.

7.2 exit Command

```
if user_input.lower() == "exit":
    print(f"\n{Fore.BLUE}🚪 Exiting Sentiment Spy. Farewell, Agent {user_name}!
🏁{Style.RESET_ALL}")
    break
```

user_input.lower(): Converts whatever the user typed to lowercase, so our command check is **case-insensitive**.

If it's "exit", we print a farewell message (with a door emoji 🚪), then **break** out of the while loop, **ending** the entire script.

7.3 reset Command

```
elif user_input.lower() == "reset":
    conversation_history.clear()
    print(f"{Fore.CYAN}🎉 All conversation history cleared!{Style.RESET_ALL}")
    continue
```

conversation_history.clear(): Removes all entries from the list, effectively resetting the conversation.

We print a message to confirm.

continue ensures we skip any further analysis logic and go directly back to prompt for new user input.

7.4 history Command

```
elif user_input.lower() == "history":
    if not conversation_history:
        print(f"{Fore.YELLOW}No conversation history yet.{Style.RESET_ALL}")
    else:
        print(f"{Fore.CYAN}📜 Conversation History:{Style.RESET_ALL}")
        for idx, (text, polarity, sentiment_type) in enumerate(conversation_history, start=1):
            if sentiment_type == "Positive":
                color = Fore.GREEN
                emoji = "😊"
            elif sentiment_type == "Negative":
                color = Fore.RED
                emoji = "😢"
            else:
                color = Fore.YELLOW
                emoji = "😐"
            print(f"{idx}. {color}{emoji} {text} "
                  f"(Polarity: {polarity:.2f}, {sentiment_type}){Style.RESET_ALL}")
    continue
```

If the user typed "history", we check whether conversation_history is empty.

If it **is empty**, we say so.

Otherwise, we **enumerate** through each entry:

enumerate(..., start=1): returns pairs like (1, (text, polarity, type)), (2, (text, polarity, type)), etc., so we can print a numbered list.

Each element is a **tuple**: (text, polarity, sentiment_type).

We use an **if-elif-else** chain to pick the color and emoji based on the sentiment_type:

Positive → Fore.GREEN, 😊

Negative → Fore.RED, 😢

Neutral → Fore.YELLOW, 😐

Finally, we format the output to show:

Message index

Color + **Emoji** + **text**

The **polarity** (rounded to 2 decimals using :.2f)

The **sentiment type**

Style.RESET_ALL again resets color so future text is not tinted.

7.5 Default Case: Sentiment Analysis

```
polarity = TextBlob(user_input).sentiment.polarity
if polarity > 0.25:
    sentiment_type = "Positive"
    color = Fore.GREEN
    emoji = "😊"
elif polarity < -0.25:
    sentiment_type = "Negative"
    color = Fore.RED
    emoji = "😢"
else:
    sentiment_type = "Neutral"
    color = Fore.YELLOW
    emoji = "😐"
```

TextBlob(user_input).sentiment.polarity: We create a **TextBlob** object with the user's text. Then we access the .sentiment attribute, which has a .polarity property in the range **-1.0** (very negative) to **+1.0** (very positive).

We define simple thresholds for classification:

> 0.25 → Positive

< -0.25 → Negative

Otherwise → Neutral (covers the range [-0.25, 0.25])

Depending on the classification, we set:

sentiment_type to either "Positive", "Negative", or "Neutral".

color to the corresponding Fore.GREEN, Fore.RED, or Fore.YELLOW.

emoji: "😊" for positive, "😢" for negative, "😐" for neutral.

7.6 Saving to Conversation History & Printing Results

python
CopyEdit

```
conversation_history.append((user_input, polarity, sentiment_type))
print(f"{color}{emoji} {sentiment_type} sentiment detected! "
    f"(Polarity: {polarity:.2f}){Style.RESET_ALL}")
```

conversation_history.append(...): We store the user's text, the calculated polarity (float), and the sentiment type string in the global list.

We then **print** a confirmation message showing:

Color (based on sentiment).

Emoji (😊, 😢, or 😐).

Sentiment type (Positive, Negative, or Neutral).

The **polarity** value, formatted to 2 decimal places :.2f.

End with **Style.RESET_ALL** to ensure subsequent console output is not color-tinted.

Overall Logic & Flow

Initialization:

The program starts by importing libraries (colorama for color, TextBlob for NLP).

Calls colorama.init() to enable colored text.

Prints a welcome message with emojis and color.

User Introduction:

Prompts for a **name**, falling back to "Mystery Agent" if none is provided.

Prints an overview of how to use the chatbot.

Main Loop:

Continually prompts the user for input.

Checks for special commands (exit, reset, history).

If none of those commands match, it performs **sentiment analysis**:

Create a **TextBlob** of the user's input.

Get the .polarity score.

Classify the text as **Positive**, **Negative**, or **Neutral** based on thresholds.

Print a **colored** and **emoji-labeled** output describing the sentiment.

Save the data to **conversation_history**.

Exiting:

If the user types **exit**, print a goodbye message, break out of the loop, and the script ends.

Key Points & Parameters

TextBlob Parameter:

We **instantiate** TextBlob with the string from user_input. There are no extra parameters in the constructor here; the default English sentiment analyzer is used.

polarity:

A **float** in the range [-1.0, +1.0].

Negative values: indicate negative sentiment; 0 is neutral; positive values indicate positive sentiment.

Colorama Attributes:

Fore: Fore.RED, Fore.GREEN, Fore.YELLOW, Fore.BLUE, Fore.CYAN, Fore.MAGENTA, etc.

Style.RESET_ALL resets **all** text styling to default.

Emojis:

Simply **Unicode characters** that most modern terminals can display. Examples used:

👋 (waving hand)

🎉 (party popper)

🕵️ (detective)

😊 (smiling face with smiling eyes)

😢 (crying face)

😐 (neutral face)

🚪 (door)

🏁 (finish flag)

📜 (scroll)

🔍 (magnifying glass)

Flow Control:

if not user_input: → no input → show error, continue.

if user_input.lower() == 'exit': break → exit loop → script ends.

elif ... == 'reset': → clear conversation, continue loop.

elif ... == 'history': → display all messages with sentiment, continue loop.

else: → perform sentiment analysis.