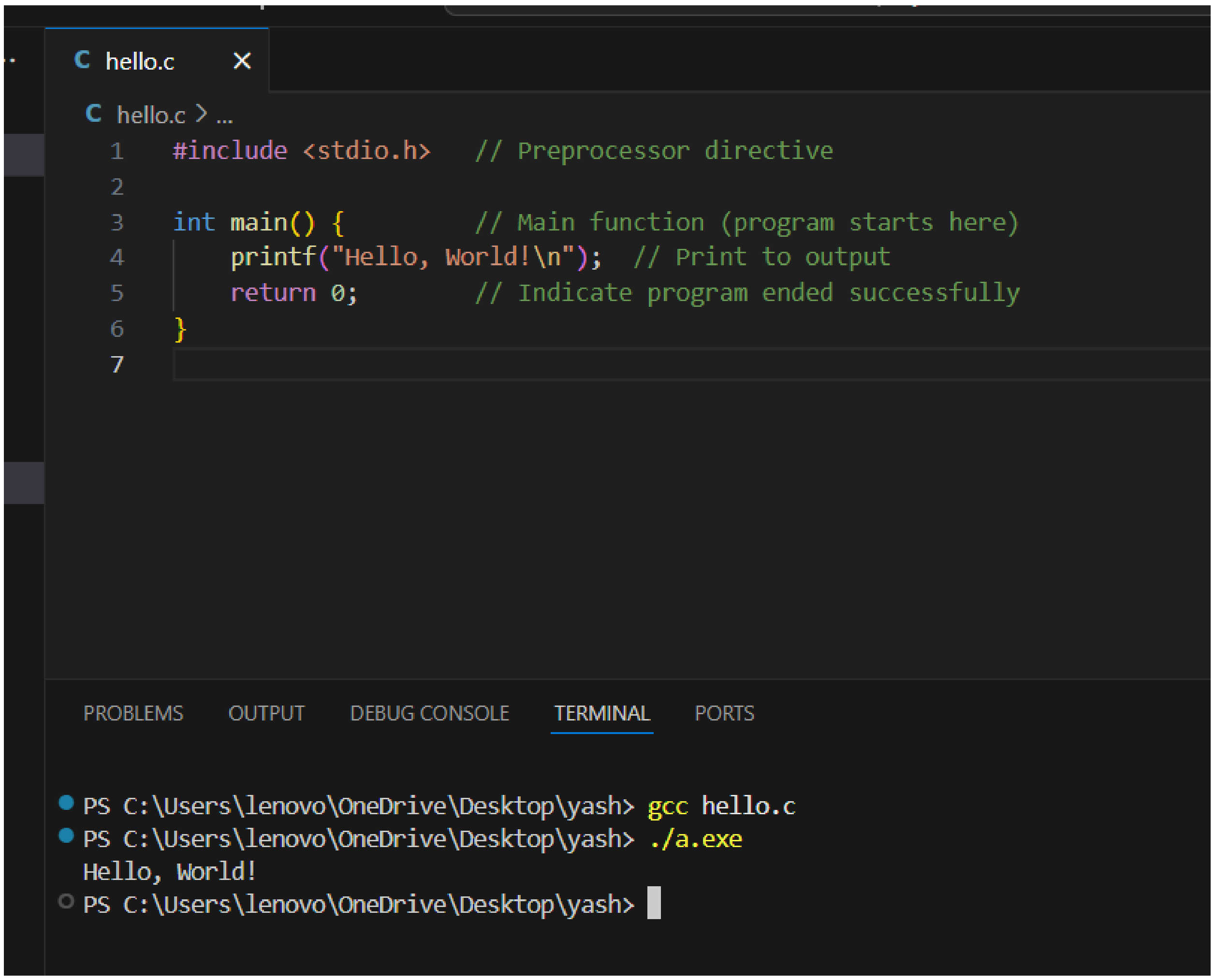
Module 1 – Overview of IT Industry

(Practical Exercises)

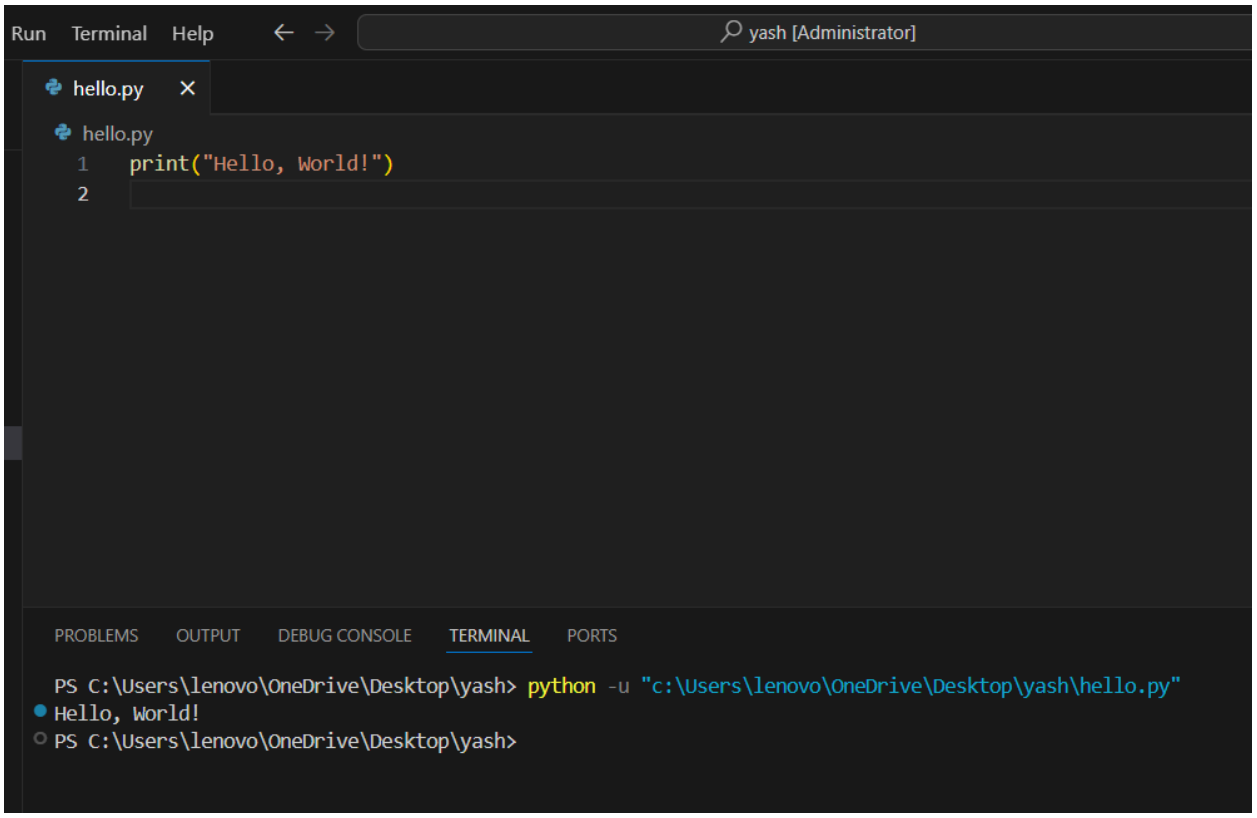
1. Write a simple 'Hello World' program in two different programming languages of your choice. Compare the structure and syntax.

Ans. =



```c
#include <stdio.h>    // Preprocessor directive

int main() {          // Main function (program starts here)
    printf("Hello, World!\n");  // Print to output
    return 0;         // Indicate program ended successfully
}

```

```
PS C:\Users\lenovo\OneDrive\Desktop\yash> gcc hello.c
PS C:\Users\lenovo\OneDrive\Desktop\yash> ./a.exe
Hello, World!
PS C:\Users\lenovo\OneDrive\Desktop\yash>
```



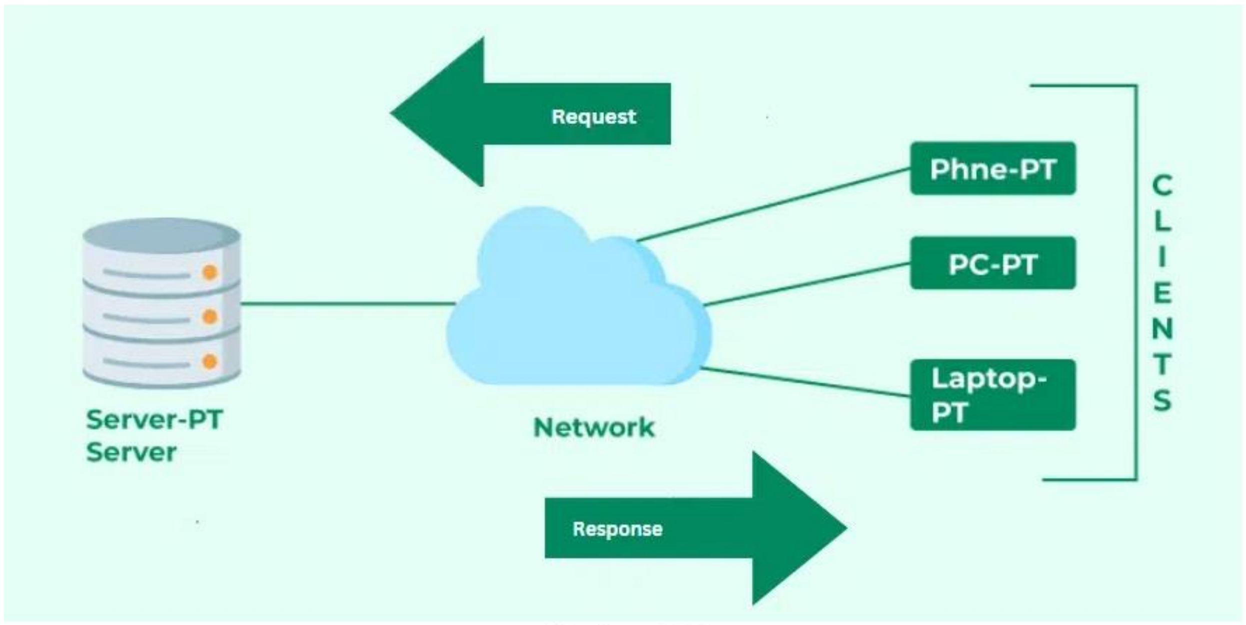
```python
print("Hello, World!")
```

```
PS C:\Users\lenovo\OneDrive\Desktop\yash> python -u "c:\Users\lenovo\OneDrive\Desktop\yash\hello.py"
Hello, World!
PS C:\Users\lenovo\OneDrive\Desktop\yash>
```

Feature	C	Python
Program Entry	Needs a main() function. Execution starts from main.	No special entry point required; execution starts from the first line.
Libraries	Must include #include <stdio.h> for input/output.	No need to import anything for basic output.
Print Statement	Uses printf() with \n for new line.	Uses print() which automatically adds a new line.
Syntax	Strict rules: semicolons ;, braces {} for blocks.	Indentation defines blocks; no semicolons needed.
Language Type	Compiled, statically typed (needs compilation before execution).	Interpreted, dynamically typed (runs directly).

2. Research and create a diagram of how data is transmitted from a client to a server over the internet.

Ans. =



Client Server Model

1. **User Enters the URL (Uniform Resource Locator):** The user types a website address (e.g., www.example.com) into the browser's address bar.

2. **DNS (Domain Name System) Lookup:** The browser sends a request to the DNS server to resolve the human-readable URL into an IP address (since computers use IP addresses to identify and connect to each other).

3. **DNS Server Resolves the Address:** The DNS server looks up the domain name and returns the IP address of the web server hosting the requested website

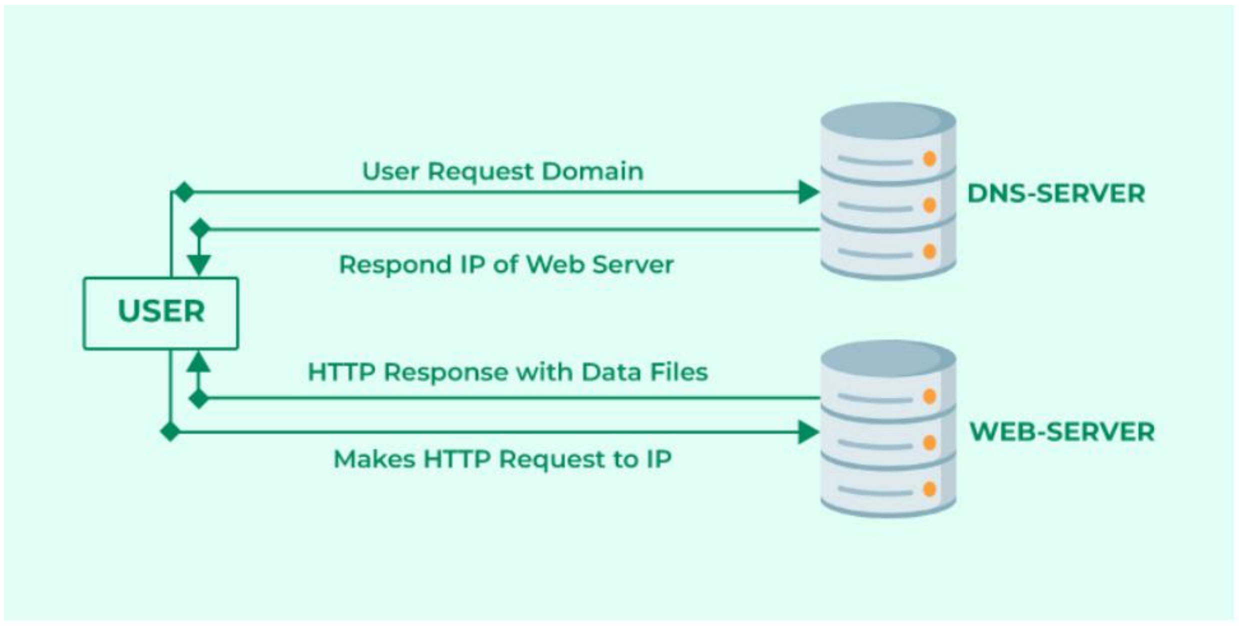
4. **Browser Sends HTTP/HTTPS Request:** The browser sends an HTTP/HTTPS request to the IP address of the web server to fetch the website's data. HTTP (HyperText Transfer Protocol) or HTTPS (the secure version) is the protocol used for communication between the browser (client) and the web server (server).

5. **Server Sends Website Files:** The server processes the request and sends the necessary website files (HTML, CSS, JavaScript, images, etc.) back to the browser.

6. **Rendering the Website:** The browser renders the files and displays the website to the user. This rendering process involves several components working together.

3. Design a simple HTTP client-server communication in any language.

Ans. =



Client Server Request and Response

Server client communication is the crucial part of any application. whenever client needs some data from server it sends a request (mostly HTTP) with require parameters and headers. Server responds to that request with response data. This type of mechanism called client to server communication where client initiates a communication with the server and server sends response through that connection only.

4. Research different types of internet connections (e.g., broadband, fiber, satellite) and list their pros and cons.

Ans. = **1. Dial-Up Internet (Legacy)**

- **Description:** Uses a telephone line to connect; very slow and outdated.
- **Pros:**
  - Very low cost.
  - Works in rural/remote areas where no other connection exists.
- **Cons:**
  - Extremely slow (56 Kbps max).
  - Cannot use phone and internet simultaneously.
  - Obsolete for modern needs (streaming, gaming, video calls).

2. DSL (Digital Subscriber Line)

- **Description:** Uses existing telephone lines but provides faster internet than dial-up.
- **Pros:**
  - Widely available in cities and small towns.
  - Affordable.

    - Always-on connection (no need to dial).
- **Cons:**
    - Speed depends on distance from ISP's central office.
    - Slower compared to fiber and cable.
    - Performance drops with high traffic.

3. Cable Internet

- **Description:** Uses coaxial TV cables to deliver internet.
- **Pros:**
    - Higher speeds than DSL.
    - Widely available in urban areas.
    - Supports streaming and gaming.
- **Cons:**
    - Shared bandwidth (speed may slow during peak hours).
    - Can be more expensive than DSL.

4. Fiber-Optic Internet

- **Description:** Uses light signals through fiber cables to provide ultra-fast internet.
- **Pros:**
    - Extremely high speeds (up to 1 Gbps or more).
    - Very reliable, low latency.
    - Great for gaming, streaming, and business use.
- **Cons:**
    - Limited availability (mostly in cities).
    - Expensive installation and subscription.

5. Satellite Internet

- **Description:** Uses satellites to provide internet, useful in remote areas.
- **Pros:**
    - Available almost anywhere (good for rural areas).
    - Doesn't need physical cables.
- **Cons:**
    - High latency (signal travels to space and back).
    - Data caps often apply.
    - Weather can disrupt service.
    - Expensive compared to speed offered.

6. Mobile Data (3G, 4G, 5G)

- **Description:** Wireless internet through cellular networks.
- **Pros:**
    - Portable and widely available.
    - Easy to set up (just need SIM card/mobile device).

    - 5G offers very high speeds and low latency.
- **Cons:**
    - Coverage varies by region.
    - Expensive if heavy data is used.
    - Speed fluctuates based on network congestion.

7. Fixed Wireless Internet

- **Description:** Internet is delivered wirelessly from a nearby tower to a receiver at your home.
- **Pros:**
    - Faster than satellite.
    - Good for rural areas where fiber/cable isn't available.
- **Cons:**
    - Requires line-of-sight with the tower.
    - Weather interference possible.
    - Slower than fiber and cable.

5. Simulate HTTP and FTP requests using command line tools (e.g., curl).

Ans. = 1. Simulating HTTP Requests with `curl`

(a) Simple HTTP GET request

➡ Fetches the HTML page from `example.com`.

(b) HTTP GET with headers

➡ `-v` enables verbose output (shows request & response headers).

(c) Sending an HTTP POST request.

➡ Sends form data (`application/x-www-form-urlencoded`) to the server.

(d) Sending JSON data

```
curl -X POST -H "Content-Type: application/json" \
  -d '{"name":"Dhanyesh","role":"student"}' \
  http://example.com/api/users
```

(e) Downloading a file

➡ Saves `file.zip` in the current directory.

2. Simulating FTP Requests with curl

(a) Listing files on an FTP server
(b) Downloading a file via FTP

(c) Uploading a file to FTP
(d) Passive mode FTP (if needed for firewalls)

6. Identify and explain three common application security vulnerabilities. Suggest possible solutions.

Ans. = 1. **SQL Injection (SQLi)**

This vulnerability allows attackers to inject malicious SQL code into input fields. This can bypass authentication and expose, modify, or delete data from the database.

Solution: Use prepared statements or parameterized queries to separate SQL code from user input. Always validate and sanitize all user-provided data.

2. Cross-Site Scripting (XSS)

XSS involves injecting malicious scripts (e.g., JavaScript) into web pages. When other users view the page, their browsers execute the script, which can lead to session hijacking, data theft, or website defacement.

Solution: Output encode all user data before displaying it. Implement a Content Security Policy (CSP) to restrict which scripts are allowed to run on your site.

3. Cross-Site Request Forgery (CSRF)

An attacker tricks a user into unknowingly submitting malicious requests while they're authenticated on another site.

Solution : Use **CSRF tokens** in forms and requests (unique per session/user) Require **re-authentication** for sensitive actions (e.g., password change) Implement **SameSite cookies** to prevent cross-origin request abuse.

7. Identify and classify 5 applications you use daily as either system software or application software.

Ans. = System Software :

(System software manages hardware and provides a platform for applications to run.)

1. **Operating System (e.g., Windows, Linux, macOS, Android, iOS)**

   - **Type:** System Software
   - **Reason:** Controls hardware, manages resources, provides the environment for all other applications.
2. **Device Drivers (e.g., Printer Driver, Graphics Driver, Wi-Fi Driver)**
   - **Type:** System Software
   - **Reason:** Enables communication between hardware devices and the operating system.

Application Software :

(Application software is designed for end-users to perform specific tasks.)

3. **Web Browser (e.g., Chrome, Firefox, Edge)**
   - **Type:** Application Software
   - **Reason:** Used to access the internet, browse websites, run web applications.
4. **Messaging App (e.g., WhatsApp, Telegram, Messenger)**
   - **Type:** Application Software
   - **Reason:** Provides communication features like chat, voice, and video calls.
5. **Office Suite (e.g., Microsoft Word, Google Docs, Excel)**
   - **Type:** Application Software
   - **Reason:** Helps in creating, editing, and managing documents, spreadsheets, and presentations.

8. Design a basic three-tier software architecture diagram for a web application.

Ans. =

```
+--------------------+       +--------------------+       +--------------------+
|  Presentation Tier| <---> |  Application Tier | <---> |     Data Tier      |
| (Client/Browser)  |       | (Business Logic)  |       | (Database Server) |
+--------------------+       +--------------------+       +--------------------+
        ^                            ^                            ^
        |                            |                            |
        |  User Interface            |  Business Logic            |  Data
Storage
        |                            |                            |
        |  (HTML, CSS, JS)           |  (Web Server, App Server) |  (SQL/NoSQL
DB)
        |                            |                            |
        +----------------------------+----------------------------+
```

Oops, something went wrong.

9. Create a case study on the functionality of the presentation, business logic, and data access layers of a given software system.

Ans. = **Case Study: Online Food Ordering System**

• Presentation Layer (User Interface Layer)

- **Definition:**
  This layer is the "face" of the system that users interact with. It handles **UI/UX, input collection, and displaying results**.
- **Example in the Food Ordering System:**
  - Mobile app or website where users browse restaurants.
  - Users select food items, add them to the cart, and place orders.
  - Displays menus, prices, delivery time, and order status.
- **Technologies Used:**
  - Frontend frameworks: **React, Angular, Flutter**
  - Markup/UI: **HTML, CSS, JavaScript**
  - APIs to fetch and display data.
  -

• Business Logic Layer (Application Layer)

- **Definition:**
  This layer contains the **rules, workflows, and decision-making processes**. It processes data received from the presentation layer and interacts with the data access layer.
- **Example in the Food Ordering System:**
  - Validates user login and session.
  - Calculates total bill including tax and discounts.
  - Assigns delivery partner based on location and availability.
  - Handles payment gateway integration.
  - Ensures business rules like "free delivery above ₹500" or "restaurant working hours".
- **Technologies Used:**
  - Programming languages: **Java, Python (Django), Node.js, .NET**
  - Business rules engines, microservices, and APIs.

• Data Access Layer (Persistence Layer)

- **Definition:**
  This layer handles **data storage and retrieval**. It communicates with the database, applies queries, and sends structured results back to the business logic layer.
- **Example in the Food Ordering System:**
  - Stores user profiles, restaurant menus, and order history.
  - Manages payment transactions and delivery records.
  - Executes queries like "fetch restaurants near user's location" or "retrieve pending orders".
- **Technologies Used:**
  - Databases: **MySQL, PostgreSQL, MongoDB**
  - ORM tools: **Hibernate, Sequelize, Entity Framework**

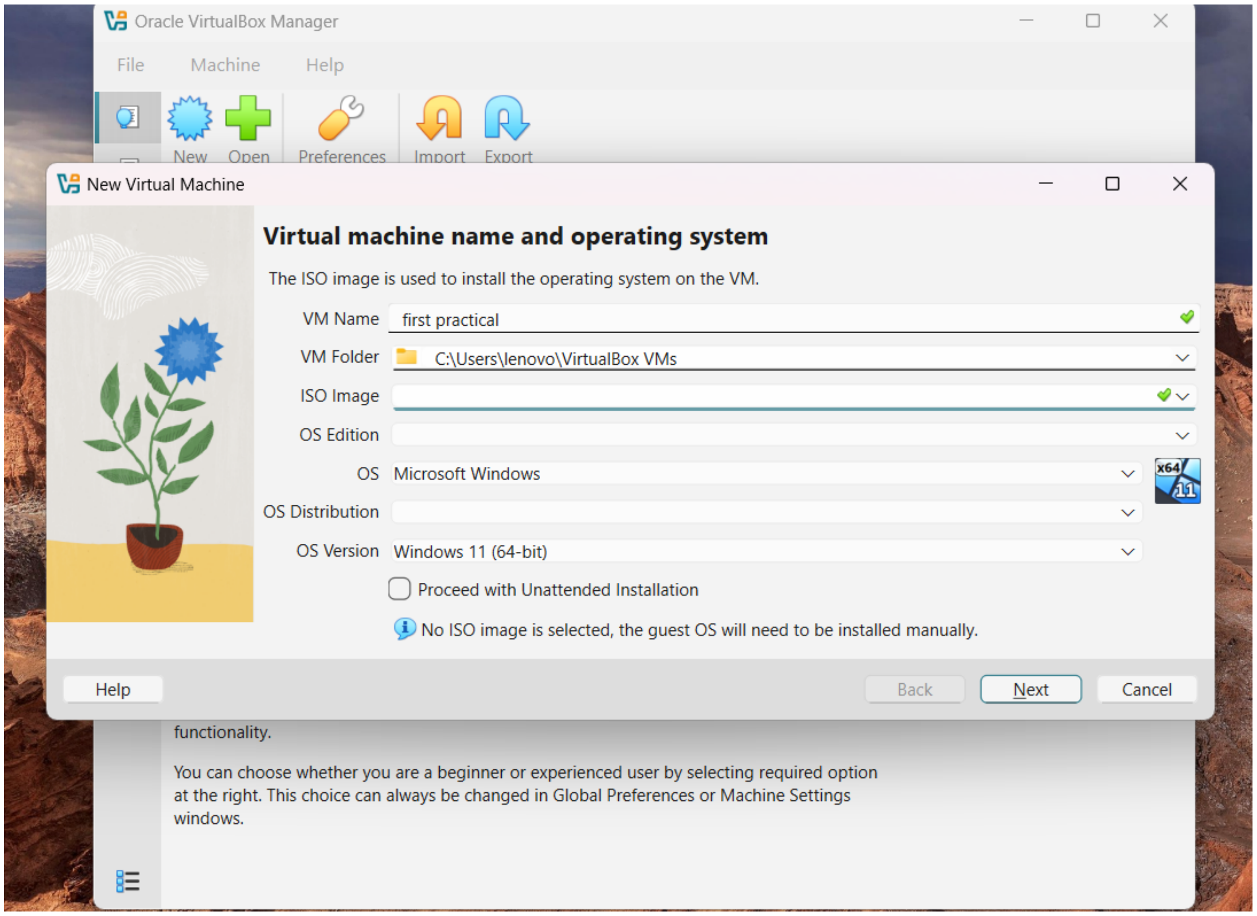
- Cloud storage for media (images of food/restaurants).

- **Interaction Between Layers**

1. **User (Presentation Layer):** Selects food items and clicks "Order Now".
2. **Business Logic Layer:** Validates order, checks restaurant availability, calculates price, and processes payment.
3. **Data Access Layer:** Saves the order details, updates restaurant stock, and retrieves delivery partner information.
4. **Back to Presentation Layer:** Displays confirmation and estimated delivery time to the user.

10. Explore different types of software environments (development, testing, production). Set up a basic environment in a virtual machine.

Ans. =



Virtual machine name and operating system

The ISO image is used to install the operating system on the VM.

VM Name: first practical

VM Folder: C:\Users\lenovo\VirtualBox VMs

ISO Image:

OS Edition:

OS: Microsoft Windows

OS Distribution:

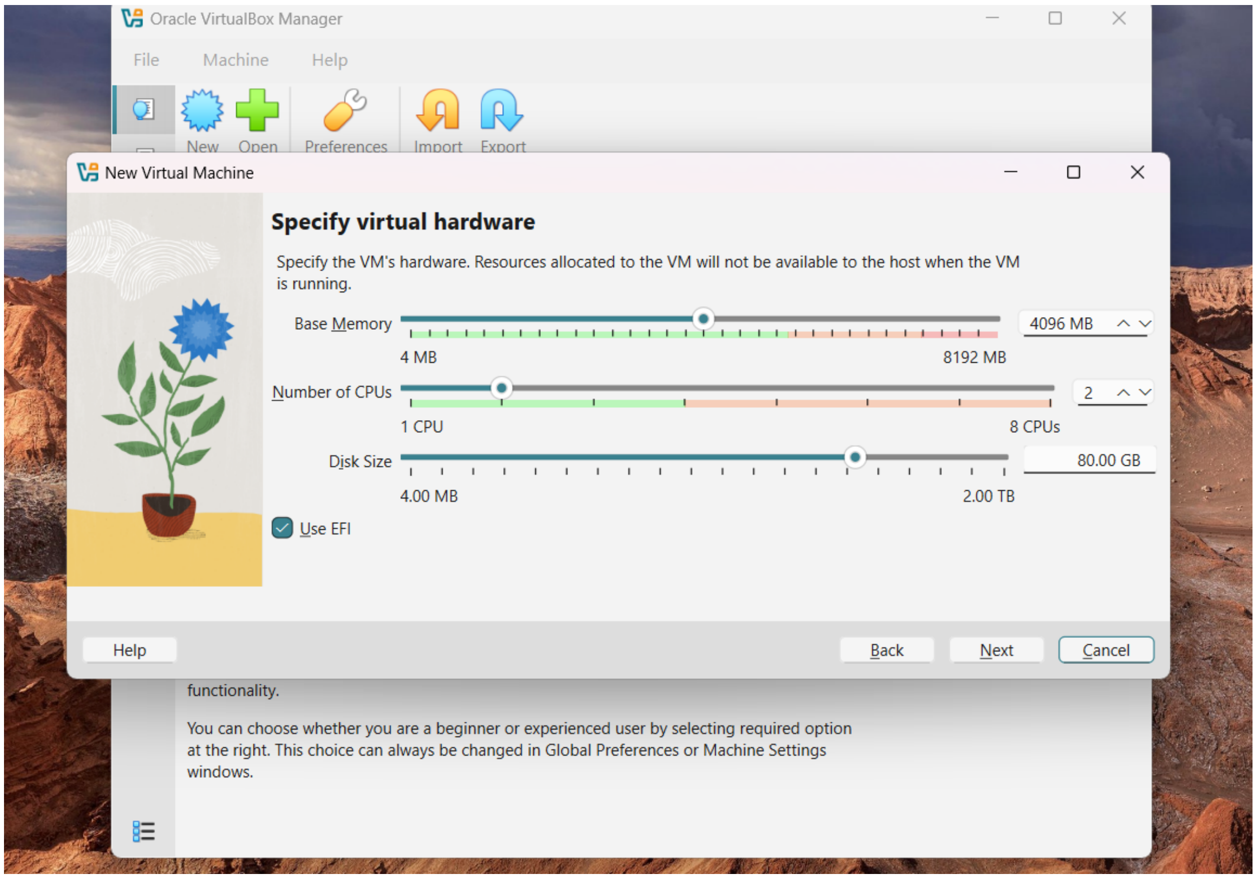
OS Version: Windows 11 (64-bit)

Proceed with Unattended Installation

No ISO image is selected, the guest OS will need to be installed manually.

functionality.

You can choose whether you are a beginner or experienced user by selecting required option at the right. This choice can always be changed in Global Preferences or Machine Settings windows.



Specify virtual hardware

Specify the VM's hardware. Resources allocated to the VM will not be available to the host when the VM is running.

Base Memory: 4096 MB (4 MB – 8192 MB)

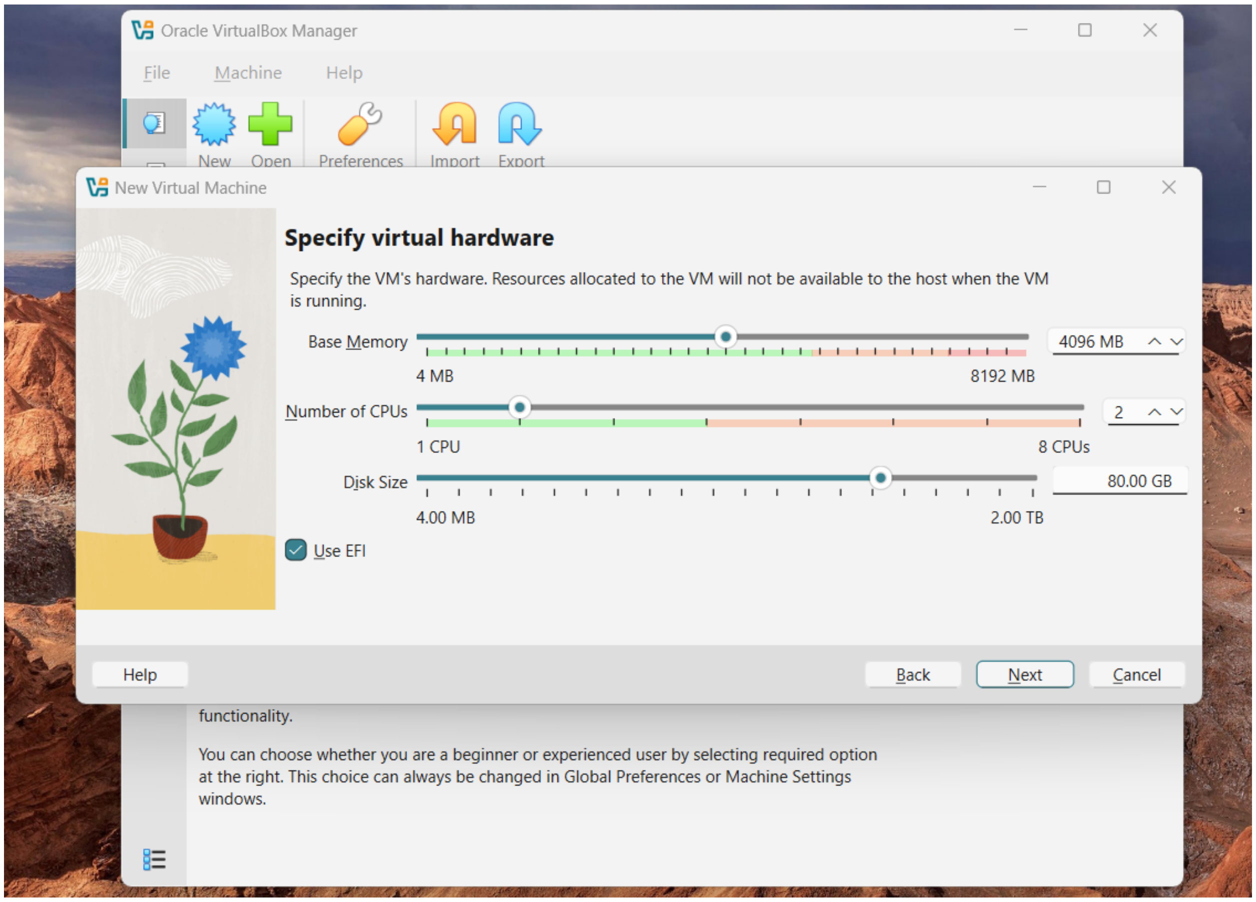
Number of CPUs: 2 (1 CPU – 8 CPUs)

Disk Size: 80.00 GB (4.00 MB – 2.00 TB)

✓ Use EFI

functionality.

You can choose whether you are a beginner or experienced user by selecting required option at the right. This choice can always be changed in Global Preferences or Machine Settings windows.



Specify virtual hardware

Specify the VM's hardware. Resources allocated to the VM will not be available to the host when the VM is running.

Base Memory 4096 MB (4 MB – 8192 MB)

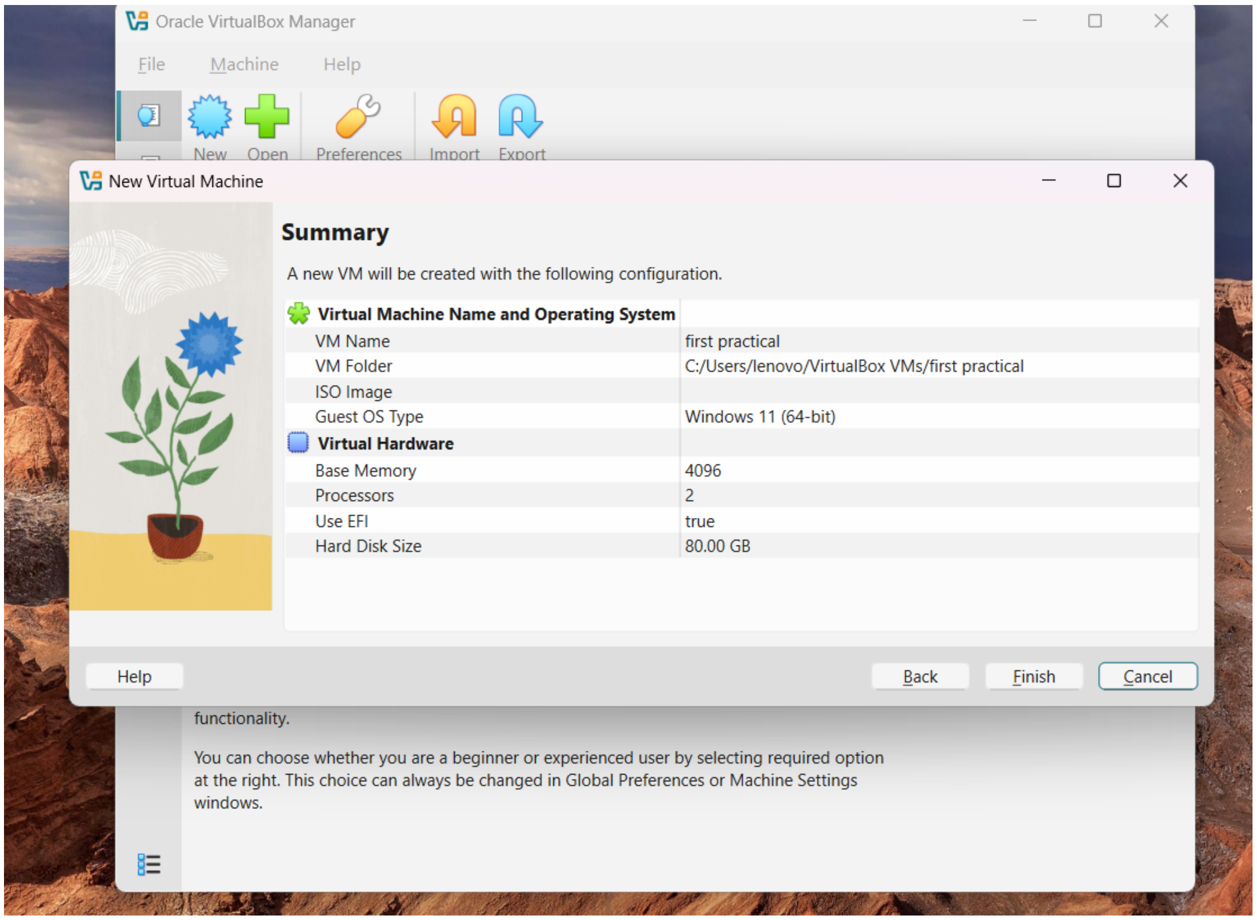
Number of CPUs 2 (1 CPU – 8 CPUs)

Disk Size 80.00 GB (4.00 MB – 2.00 TB)

[✓] Use EFI

functionality.

You can choose whether you are a beginner or experienced user by selecting required option at the right. This choice can always be changed in Global Preferences or Machine Settings windows.



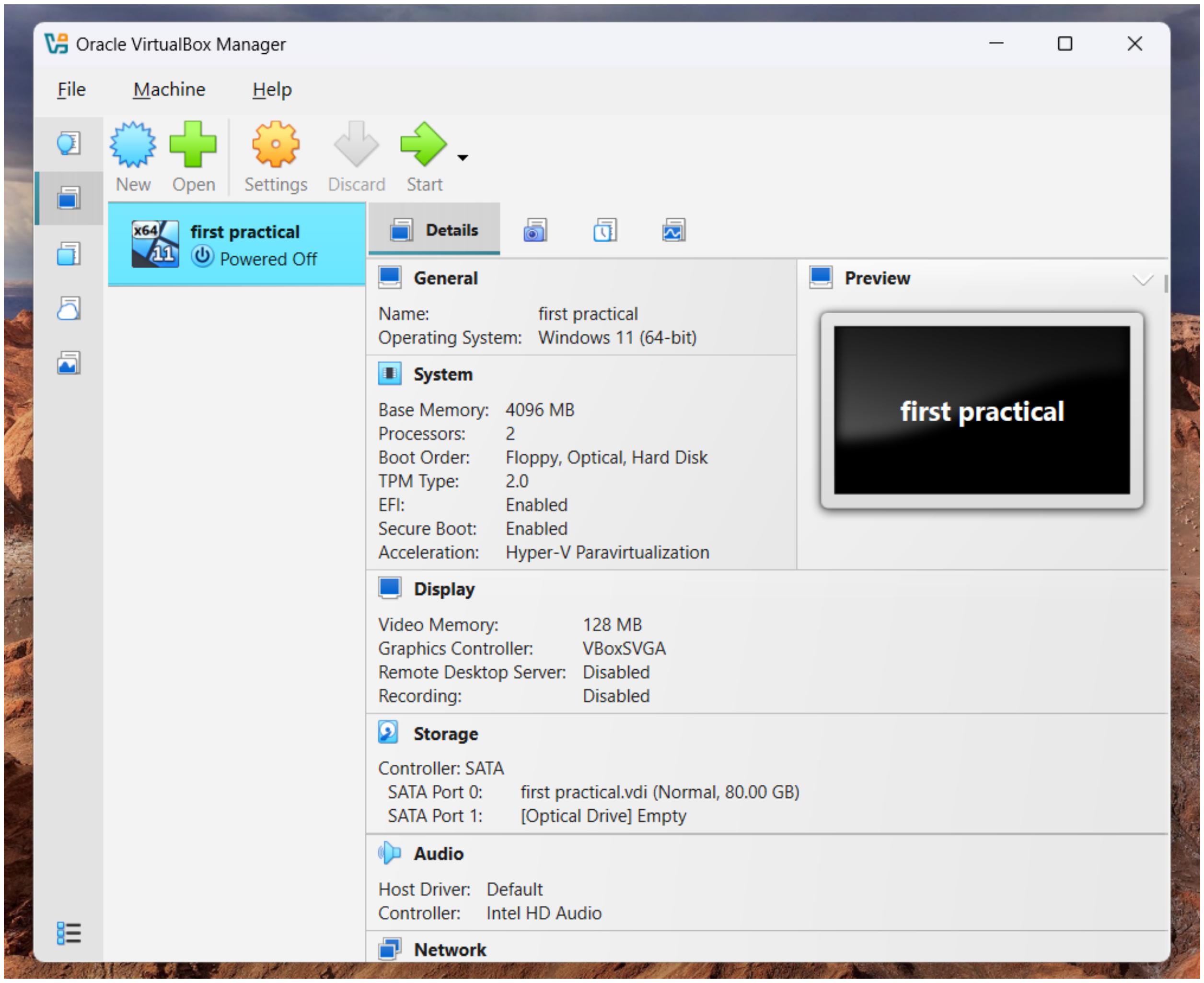
Summary

A new VM will be created with the following configuration.

Virtual Machine Name and Operating System	
VM Name	first practical
VM Folder	C:/Users/lenovo/VirtualBox VMs/first practical
ISO Image	
Guest OS Type	Windows 11 (64-bit)
Virtual Hardware	
Base Memory	4096
Processors	2
Use EFI	true
Hard Disk Size	80.00 GB

functionality.

You can choose whether you are a beginner or experienced user by selecting required option at the right. This choice can always be changed in Global Preferences or Machine Settings windows.



General
Name: first practical
Operating System: Windows 11 (64-bit)

System
Base Memory: 4096 MB
Processors: 2
Boot Order: Floppy, Optical, Hard Disk
TPM Type: 2.0
EFI: Enabled
Secure Boot: Enabled
Acceleration: Hyper-V Paravirtualization

Display
Video Memory: 128 MB
Graphics Controller: VBoxSVGA
Remote Desktop Server: Disabled
Recording: Disabled

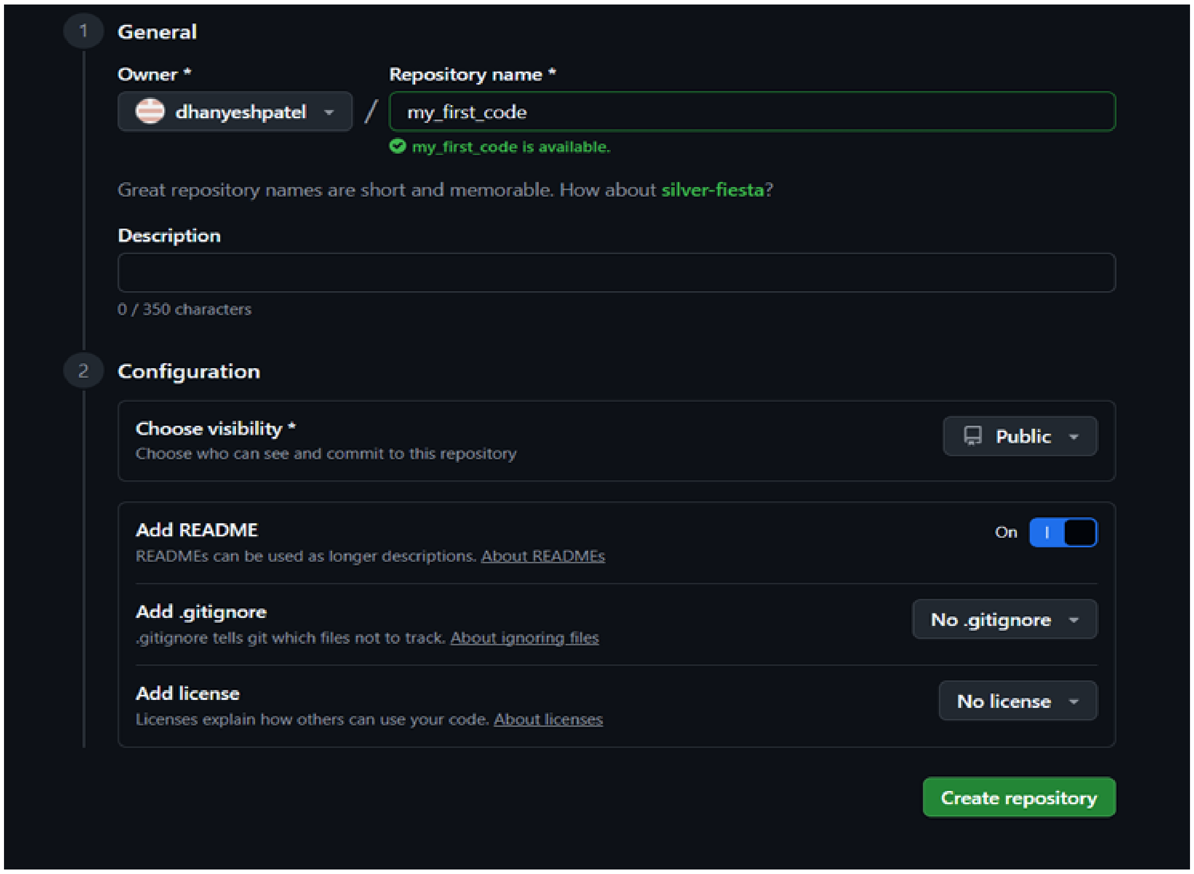
Storage
Controller: SATA
SATA Port 0: first practical.vdi (Normal, 80.00 GB)
SATA Port 1: [Optical Drive] Empty

Audio
Host Driver: Default
Controller: Intel HD Audio

Network

11. Write and upload your first source code file to Github.

Ans. = **Creating a New Repository :**



1 General

Owner * dhanyeshpatel / Repository name * my_first_code

my_first_code is available.

Great repository names are short and memorable. How about silver-fiesta?

Description

0 / 350 characters

2 Configuration

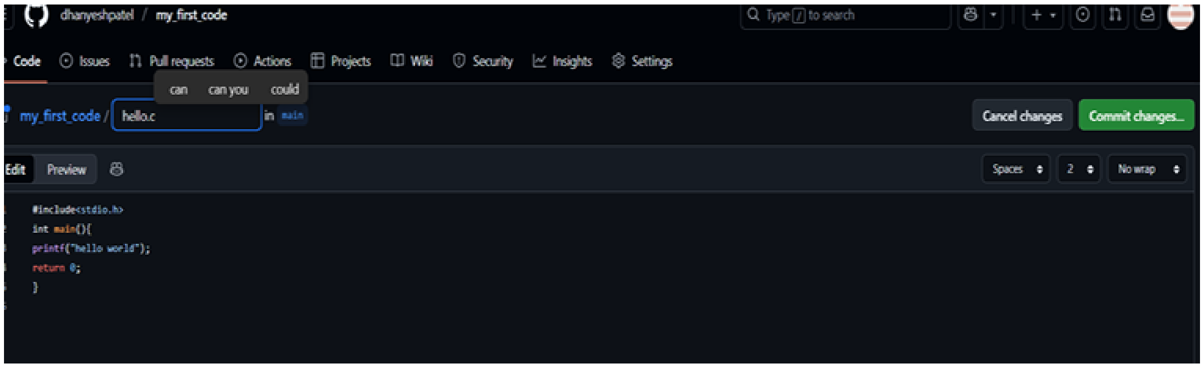
Choose visibility * — Choose who can see and commit to this repository — Public

Add README — READMEs can be used as longer descriptions. About READMEs — On

Add .gitignore — .gitignore tells git which files not to track. About ignoring files — No .gitignore

Add license — Licenses explain how others can use your code. About licenses — No license

- **Adding Source Code File :**

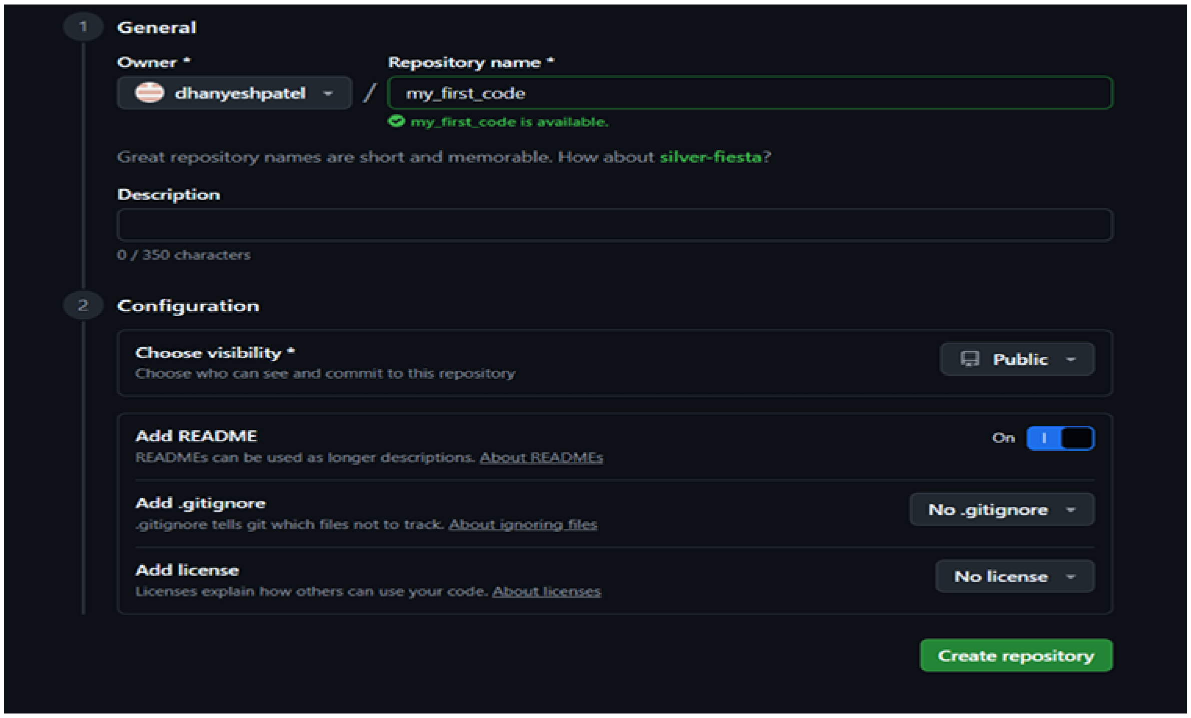


dhanyeshpatel / my_first_code
Code Issues Pull requests Actions Projects Wiki Security Insights Settings
Cancel changes
Commit changes...
Edit Preview
Spaces 2 No wrap

my_first_code / hello.c in main

```
#include<stdio.h>
int main(){
printf("hello world");
return 0;
}
```

- **Commiting new file :-**



1 General

Owner * dhanyeshpatel / Repository name * my_first_code

my_first_code is available.

Great repository names are short and memorable. How about silver-fiesta?

Description

0 / 350 characters

2 Configuration

Choose visibility * Public

Choose who can see and commit to this repository

Add README On

READMEs can be used as longer descriptions. About READMEs

Add .gitignore No .gitignore

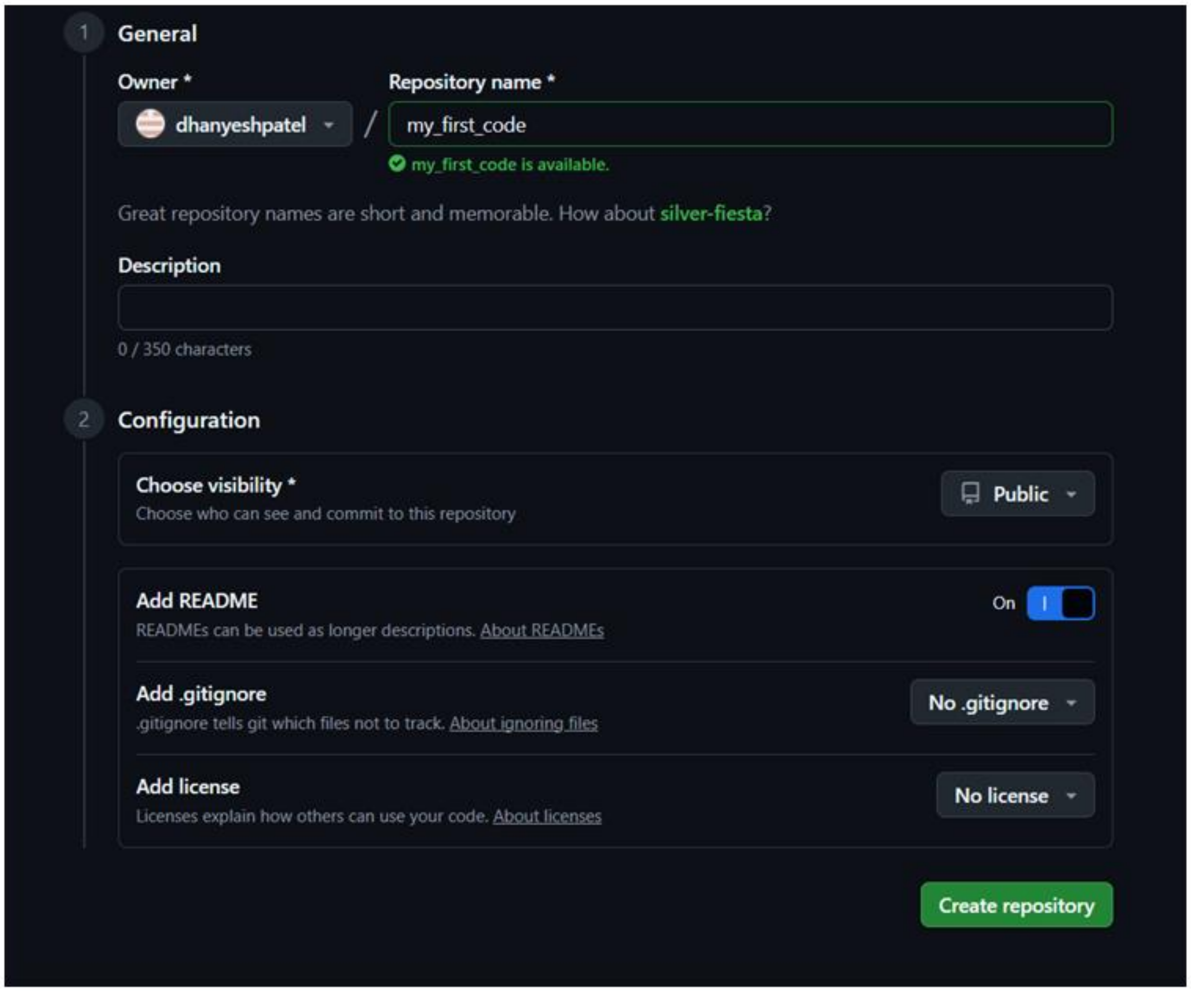
.gitignore tells git which files not to track. About ignoring files

Add license No license

Licenses explain how others can use your code. About licenses

12. Create a Github repository and document how to commit and push code changes.

Ans. = **Creating a New Repository :**



1 General

Owner * dhanyeshpatel / Repository name * my_first_code

my_first_code is available.

Great repository names are short and memorable. How about silver-fiesta?

Description

0 / 350 characters

2 Configuration

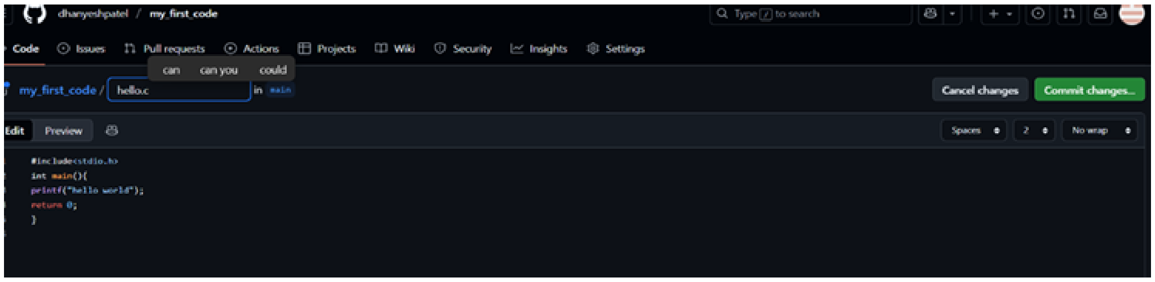
Choose visibility * — Public
Choose who can see and commit to this repository

Add README — On
READMEs can be used as longer descriptions. About READMEs

Add .gitignore — No .gitignore
.gitignore tells git which files not to track. About ignoring files

Add license — No license
Licenses explain how others can use your code. About licenses

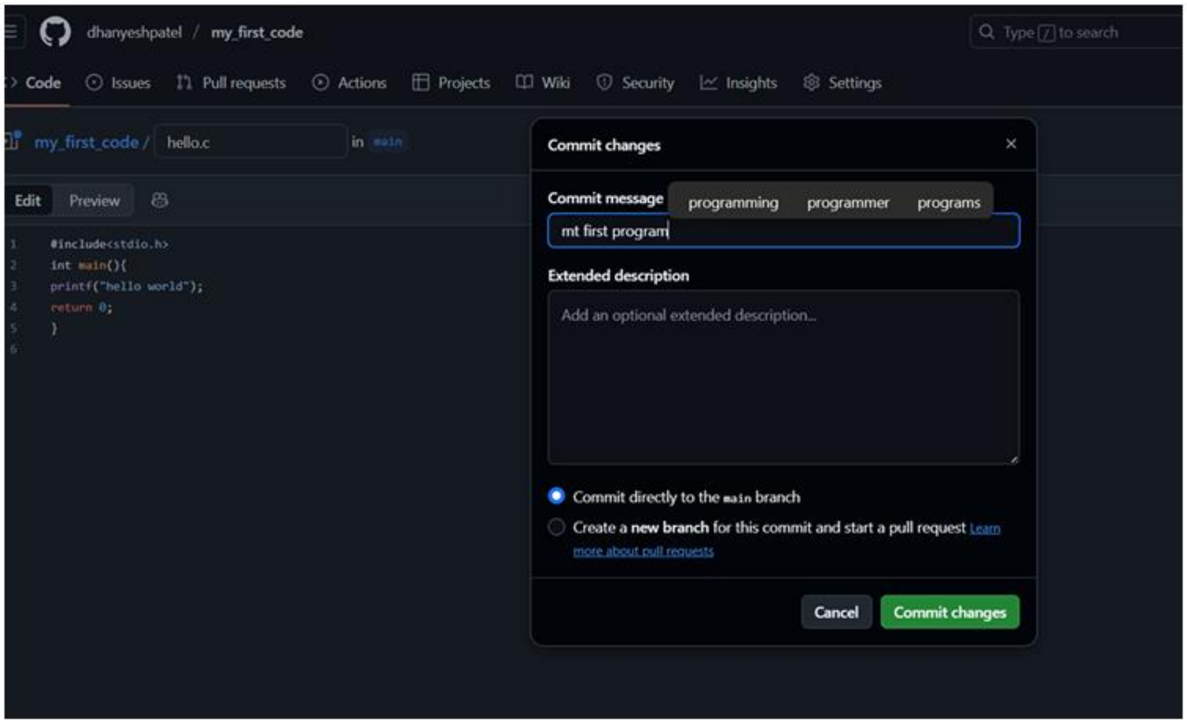
- **Adding Source Code File :**



my_first_code / hello.c in main

```
#include<stdio.h>
int main(){
printf("hello world");
return 0;
}
```

- Commiting new file :


dhanyeshpatel / my_first_code
Type / to search
Code Issues Pull requests Actions Projects Wiki Security Insights Settings
my_first_code / hello.c in main
Edit Preview

```
#include<stdio.h>
int main(){
printf("hello world");
return 0;
}
```

Commit changes
Commit message
programming programmer programs

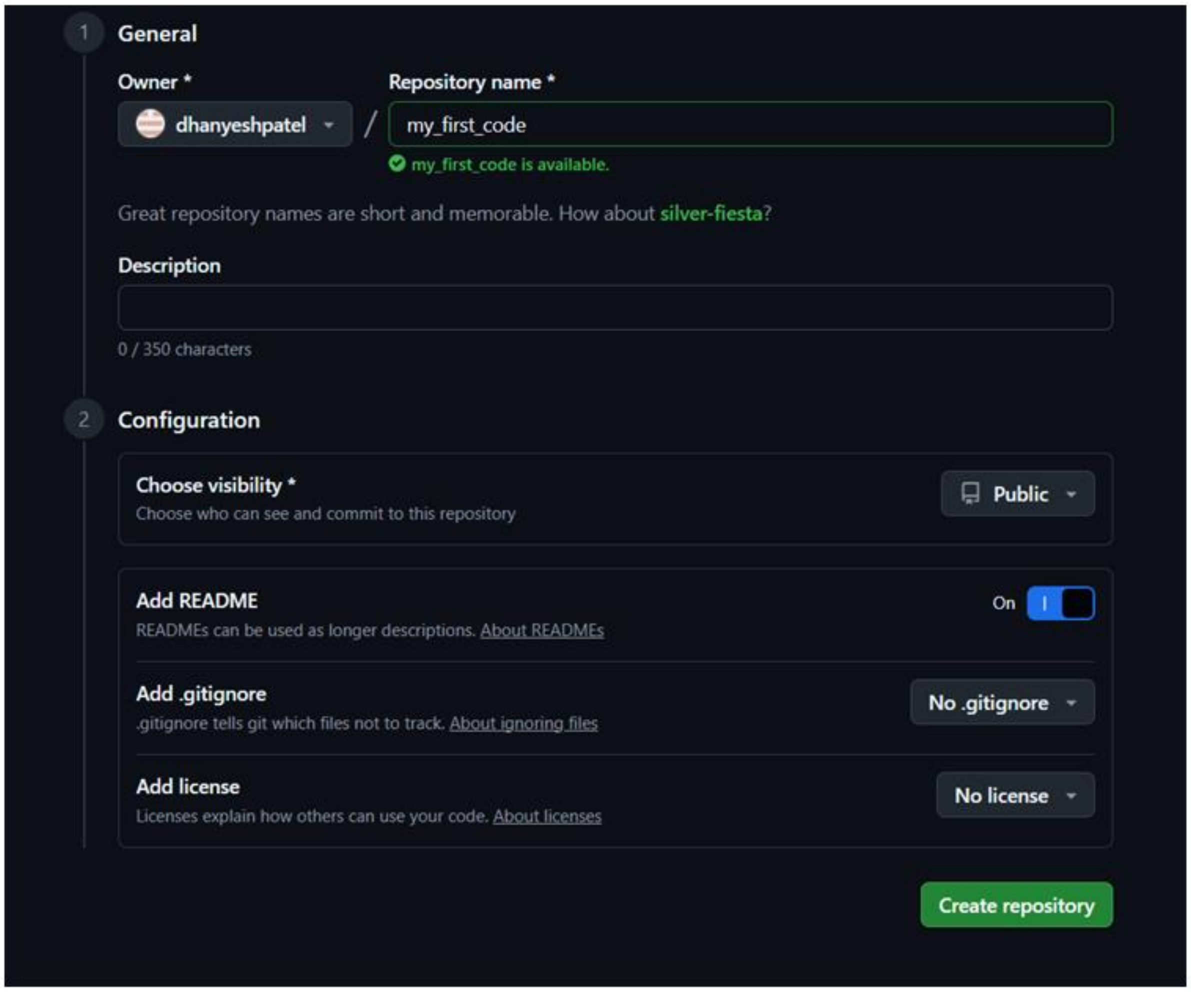
mt first program

Extended description
Add an optional extended description...

(●) Commit directly to the main branch
( ) Create a **new branch** for this commit and start a pull request Learn more about pull requests

13. Create a student account on Github and collaborate on a small project with a classmate.

Ans. = **Github Repository (Project File) :**



1 General

Owner * dhanyeshpatel / Repository name * my_first_code

my_first_code is available.

Great repository names are short and memorable. How about silver-fiesta?

Description

0 / 350 characters

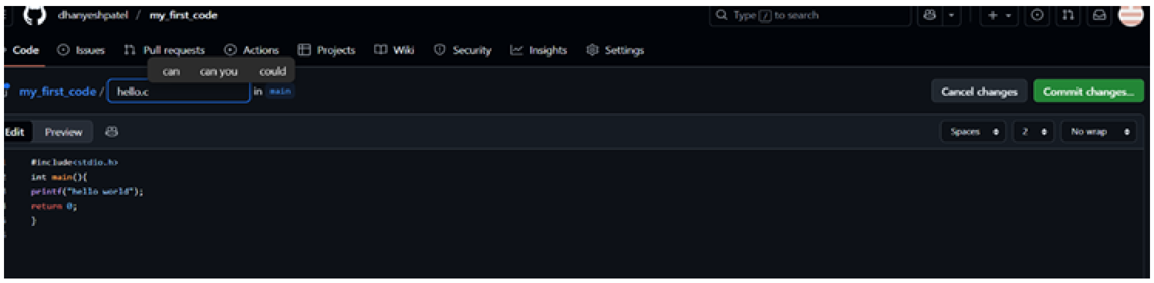
2 Configuration

Choose visibility * — Choose who can see and commit to this repository: Public

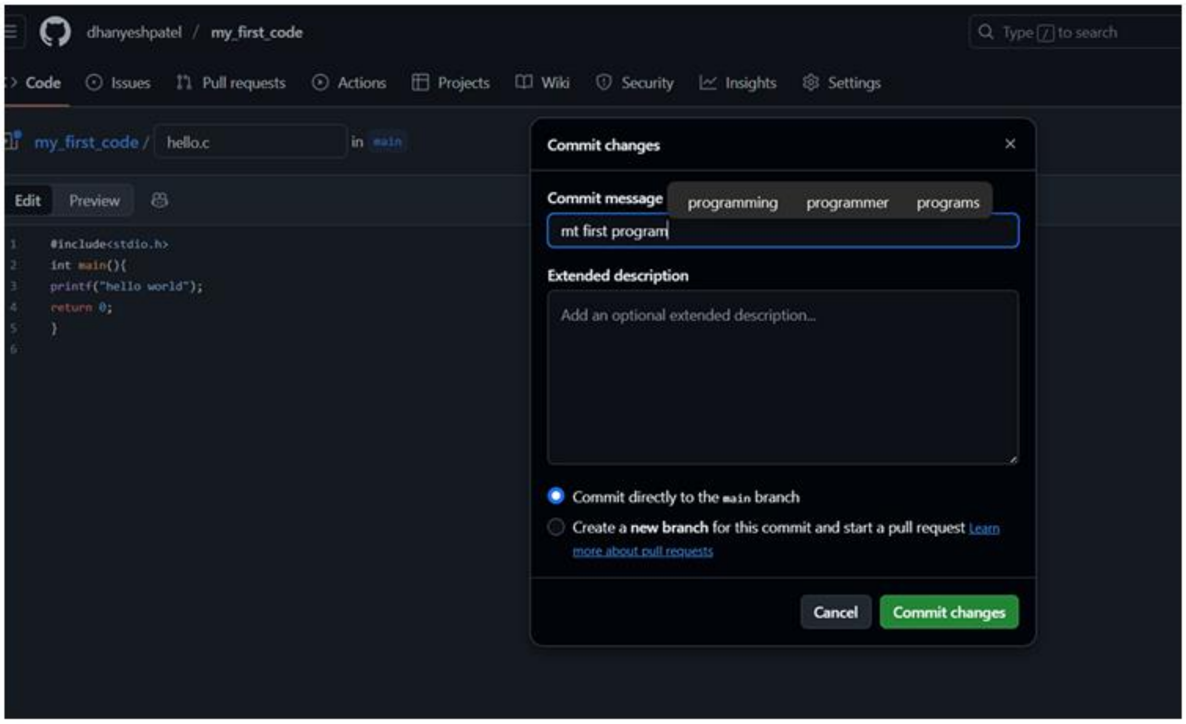
Add README — READMEs can be used as longer descriptions. About READMEs: On

Add .gitignore — .gitignore tells git which files not to track. About ignoring files: No .gitignore

Add license — Licenses explain how others can use your code. About licenses: No license



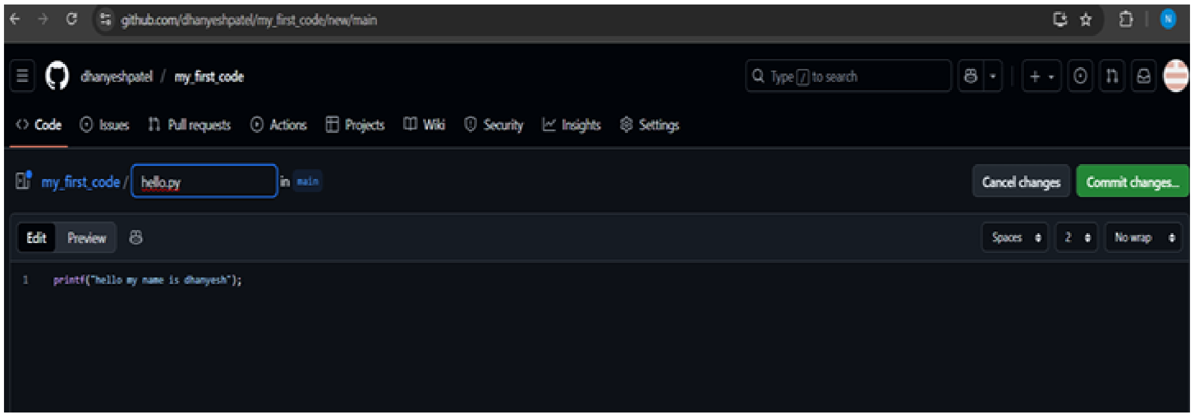
```
#include<stdio.h>
int main(){
printf("hello world");
return 0;
}
```


dhanyeshpatel / my_first_code
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
my_first_code / hello.c in main
Edit
Preview
Commit changes
Commit message
programming
programmer
programs
Extended description
Add an optional extended description...
Commit directly to the main branch
Create a new branch for this commit and start a pull request Learn more about pull requests
Cancel
Commit changes

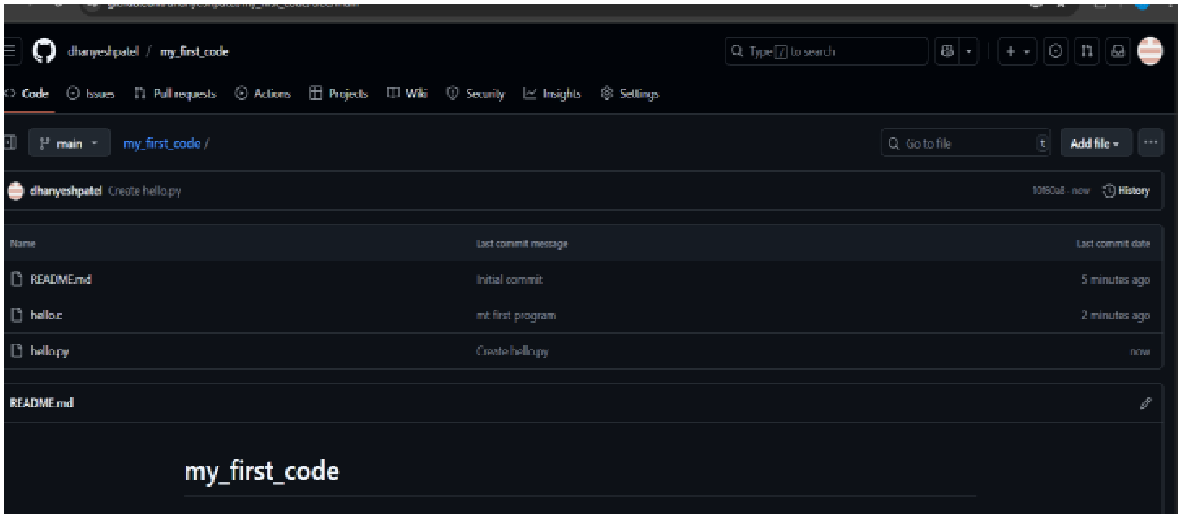
```
#include<stdio.h>
int main(){
printf("hello world");
return 0;
}
```

Commit message: mt first program

- Collaborator's work (Classmate) :



```
printf("hello my name is dhanyesh");
```


dhanyeshpatel / my_first_code

main · my_first_code /

dhanyeshpatel Create hello.py · 10f60a8 · now · History

Name	Last commit message	Last commit date
README.md	Initial commit	5 minutes ago
hello.c	mt first program	2 minutes ago
hello.py	Create hello.py	now

README.md

my_first_code

14. Create a list of software you use regularly and classify them into the following categories: system, application, and utility software.

Ans. =

- System Software

- **Windows / Linux / macOS** → Operating Systems
- **Android / iOS** → Mobile Operating Systems
- **Device Drivers** → Graphics driver, Printer driver, Wi-Fi driver

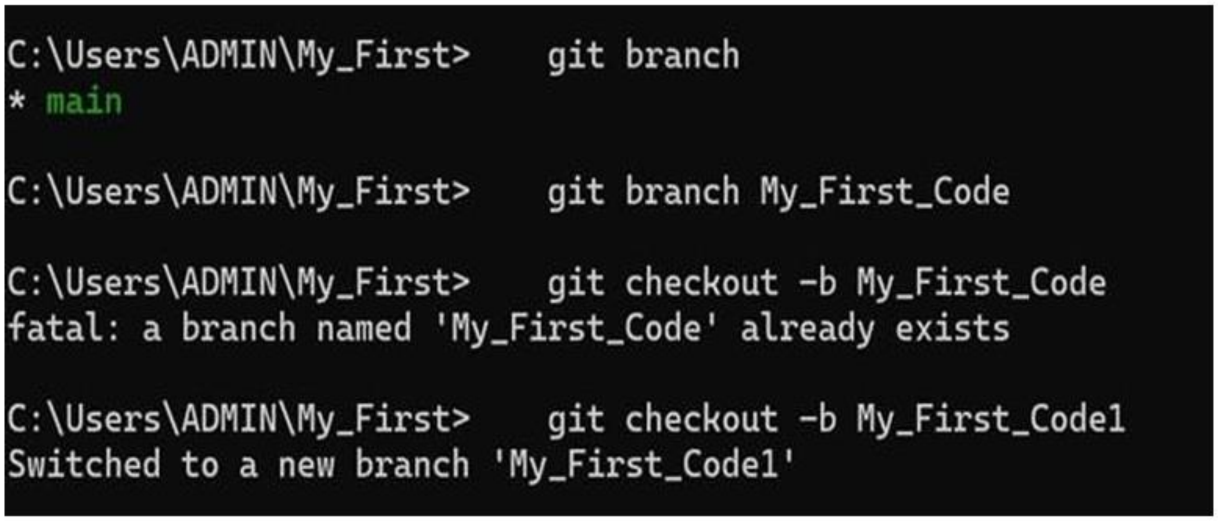
- Application Software

- **Google Chrome / Firefox / Edge** → Web Browsers
- **MS Word, Excel, PowerPoint / Google Docs & Sheets** → Office Applications
- **WhatsApp / Telegram / Gmail** → Communication Apps
- **YouTube / Spotify** → Entertainment Apps
- **Zoom / MS Teams** → Video Conferencing Tools

- Utility Software
- **Antivirus Software (e.g., Avast, Windows Defender)** → Security
- **WinRAR / 7-Zip** → File Compression
- **CCleaner** → System Cleanup & Optimization
- **Backup Tools (e.g., Google Drive, OneDrive)** → Data Backup
- **Task Manager / Resource Monitor** → System Monitoring

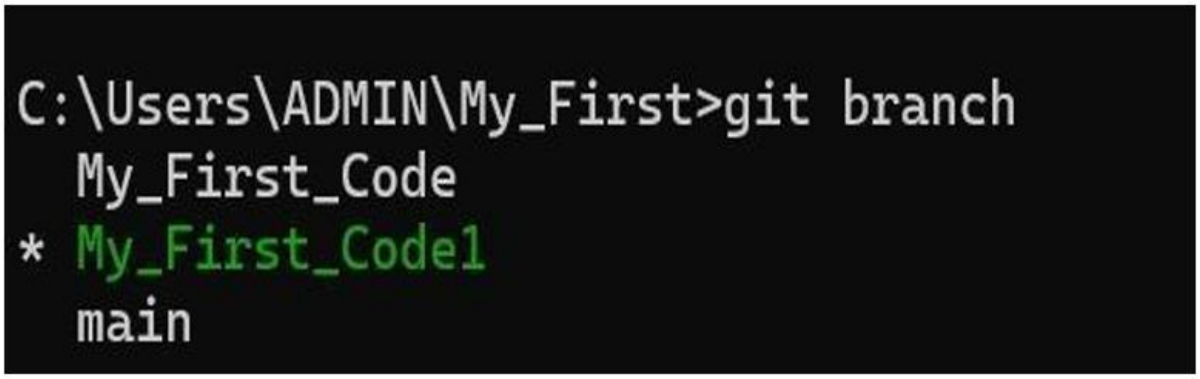
15. Follow a GIT tutorial to practice cloning, branching, and merging repositories.

Ans. = **Cloning a Repository :**



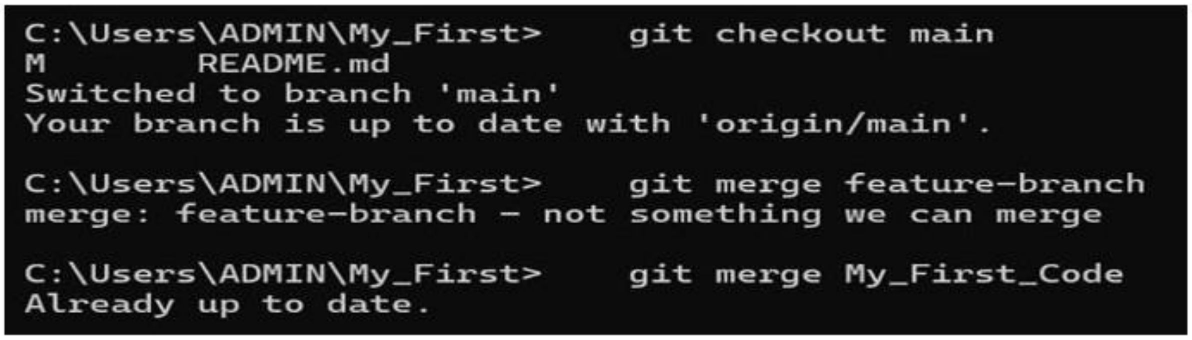
```
C:\Users\ADMIN\My_First>    git branch
* main

C:\Users\ADMIN\My_First>    git branch My_First_Code

C:\Users\ADMIN\My_First>    git checkout -b My_First_Code
fatal: a branch named 'My_First_Code' already exists

C:\Users\ADMIN\My_First>    git checkout -b My_First_Code1
Switched to a new branch 'My_First_Code1'
```

Branching a Repository :



```
C:\Users\ADMIN\My_First>git branch
  My_First_Code
* My_First_Code1
  main
```

Merging a Repository :



```
C:\Users\ADMIN\My_First>    git checkout main
M       README.md
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

C:\Users\ADMIN\My_First>    git merge feature-branch
merge: feature-branch - not something we can merge

C:\Users\ADMIN\My_First>    git merge My_First_Code
Already up to date.
```

16. Write a report on the various types of application software and how they improve productivity.

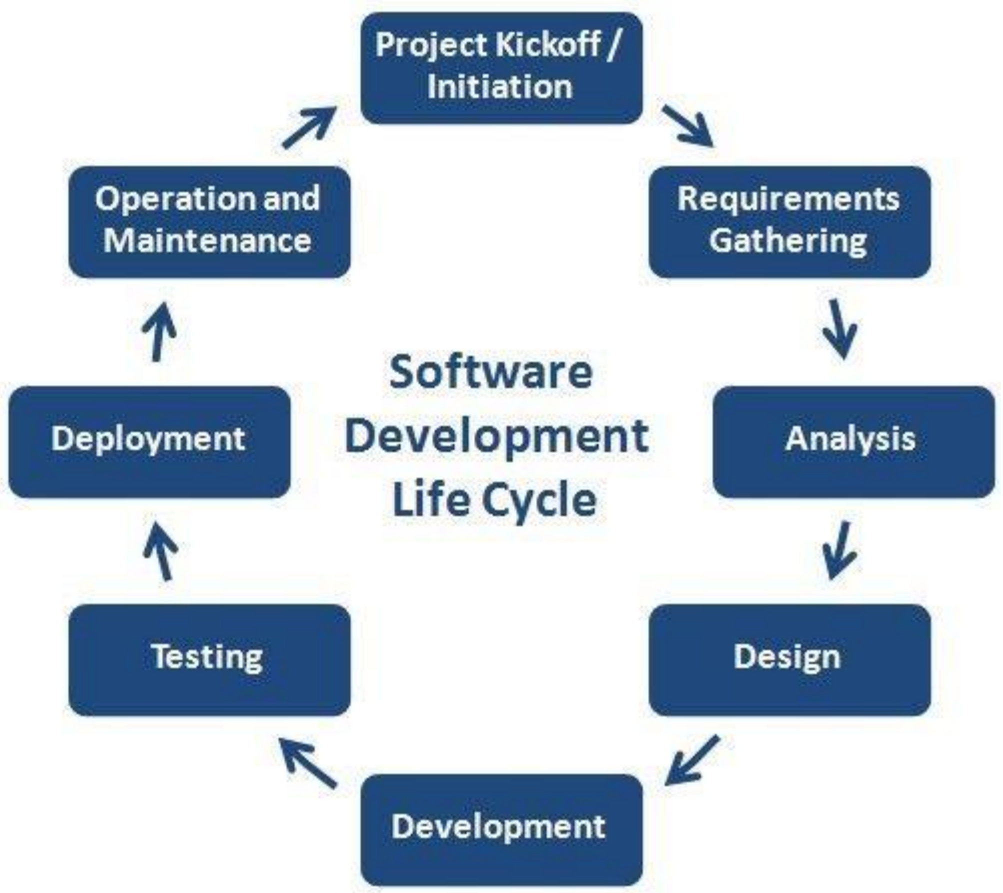
Ans. = **Report on Application Software and Productivity**

Application software refers to computer programs designed to help users perform specific tasks. Unlike system software, which manages hardware, application software focuses on improving efficiency, communication, creativity, and decision-making. With the growing reliance on technology, application software plays a vital role in personal, educational, and business productivity.

- **Types of Application Software**

1. Word Processing Software
2. Spreadsheet Software
3. Database Management Software (DBMS)
4. Presentation Software
5. Communication and Collaboration Software
6. Graphics and Multimedia Software
7. Enterprise Software (Business Applications)
8. Educational and E-Learning Software

17. Create a flowchart representing the Software Development Life Cycle (SDLC).


Project Kickoff / Initiation
Requirements Gathering
Analysis
Design
Development
Testing
Deployment
Operation and Maintenance
Software Development Life Cycle

Ans. =

18. Write a requirement specification for a simple library management system.

Ans. = **Software Requirement Specification (SRS)**

for Library Management System =

1. Introduction

1.1 Purpose

The purpose of this Library Management System is to automate library operations such as managing books, issuing and returning books, and keeping track of members and fines. This system reduces manual effort and improves efficiency.

1.2 Scope

- The system will allow **librarians** to add, update, delete, and search books.
- **Members** can search for books, borrow and return them.
- The system will manage due dates and calculate fines for late returns.

- The system should provide reports such as issued books, available books, and member history.

1.3 Users

- **Admin/Librarian** – manages the library.
- **Member/Student** – borrows and returns books.

2. Functional Requirements

2.1 Book Management

- The system shall allow the librarian to **add new books** with details (Title, Author, ISBN, Category, Availability).
- The system shall allow the librarian to **update or remove book records**.
- The system shall allow members to **search books** by title, author, or category.

2.2 Member Management

- The system shall allow the librarian to **register new members** with details (Name, ID, Contact Info).
- The system shall allow updating and deleting member records.

2.3 Borrowing and Returning Books

- The system shall allow members to **borrow available books**.
- The system shall record the **issue date and due date**.
- The system shall allow members to **return books**, updating availability.
- The system shall calculate **fines for late returns** (e.g., ₹5 per day).

2.4 Reports

- The system shall generate reports of:
  - List of available books
  - List of issued books and due dates
  - Member borrowing history

3. Non-Functional Requirements

3.1 Performance

- The system should handle at least **1000 books** and **500 members** efficiently.

3.2 Usability

- The interface should be simple, user-friendly, and accessible to non-technical users.

3.3 Reliability

- Data should not be lost during crashes; system must provide backup options.

3.4 Security

- Only librarians can add/remove books and members.
- Members must log in to borrow or return books.

3.5 Portability

- The system should run on Windows, Linux, or a web-based environment.

4. System Models

4.1 Use Case Examples

1. **Borrow Book** – Member selects a book → System checks availability → If available, book is issued.
2. **Return Book** – Member returns book → System updates record → Fine calculated if overdue.
3. **Add Book** – Librarian enters details → Book is stored in the database.

19. Perform a functional analysis for an online shopping system.

Ans. = **Functional Analysis of an Online Shopping System**

1. Introduction

The online shopping system allows customers to browse products, add items to their cart, place orders, and make payments. Sellers can manage their products, and admins oversee the entire system.

2. Stakeholders & Users

- **Customer/User** → Browses, searches, purchases products.
- **Seller/Vendor** → Manages product listings and inventory.
- **Admin** → Maintains the system, handles disputes, and monitors activities.

3. Core Functional Requirements

A. User Management

- Register new users (customers and sellers).
- Login/Logout functionality with authentication.
- Profile management (update name, email, password, address).

B. Product Management

- Sellers can add new products with details (name, description, price, stock, category).
- Update and delete products.
- Customers can browse and search products by category, price, or name.

C. Shopping Cart & Wishlist

- Customers can add/remove items in the cart.
- Update quantity of items in the cart.
- Save products to wishlist for later.

D. Order Management

- Customers can place an order.
- System generates order ID and stores order details.
- Order tracking (processing, shipped, delivered, cancelled).
- Return/refund functionality.

E. Payment & Checkout

- Multiple payment options (credit/debit card, UPI, net banking, COD).
- Secure transaction handling.
- Payment confirmation and invoice generation.

F. Search & Recommendation

- Search by product name, category, or filters.
- Recommendation engine (e.g., "Customers also bought").

G. Review & Rating System

- Customers can leave reviews and ratings for purchased products.
- Sellers can respond to feedback.

H. Admin Functions

- Manage users (activate/deactivate accounts).
- Manage product categories.
- Monitor transactions and generate reports.
- Handle customer complaints and disputes.

4. Non-Functional Requirements

- **Performance:** System should support thousands of users concurrently.
- **Security:** Secure login (encryption, OTP, SSL for payments).
- **Scalability:** Should handle growing product listings and users.
- **Reliability:** Ensure data integrity and backup.
- **Usability:** Easy-to-use interface for both customers and sellers.

5. Functional Flow (Example Use Case: Customer Purchase)

1. Customer logs in.
2. Browses products and adds items to cart.
3. Proceeds to checkout, enters shipping details.
4. Chooses payment method and confirms order.
5. System records order, reduces stock, and generates invoice.
6. Customer tracks delivery until completion.

20. Design a basic system architecture for a food delivery app.

Ans. = **Basic System Architecture – Food Delivery App**

1. Main Components

1. **Users (Customers)** – Browse restaurants, order food, make payments.
2. **Restaurants** – Manage menus, accept/reject orders, update availability.
3. **Delivery Partners** – Accept delivery requests, navigate to restaurant and customer.
4. **Admin** – Manages system, monitors transactions, resolves issues.

2. Layered Architecture

A. Presentation Layer (Frontend)

- **Mobile App / Website (for Customers, Restaurants, Delivery Agents, Admin)**
  - Customer: Browse food, order, pay, track delivery.
  - Restaurant: Manage menu, order dashboard.
  - Delivery Partner: Accept orders, GPS navigation.
  - Admin: Dashboard to manage users, restaurants, and orders.

- **Technologies:** React Native / Flutter (mobile), React / Angular (web).

B. Application Layer (Backend / Business Logic)

- **Modules:**
  - **User Management:** Login, registration, profile, wallet.
  - **Restaurant Management:** Menu updates, stock, working hours.
  - **Order Management:** Place order, track status, cancellation, refunds.
  - **Delivery Management:** Assign delivery partner, route optimization.
  - **Payment Gateway Integration:** UPI, credit/debit cards, wallets.
  - **Notification System:** SMS, email, push notifications.
- **Technologies:** Node.js, Django, Java (Spring Boot), or .NET.

C. Data Layer (Database & Storage)

- **Databases:**
  - SQL (PostgreSQL/MySQL) for structured data (users, orders, payments).
  - NoSQL (MongoDB/Redis) for fast lookups (real-time order status, caching).
- **Storage:** Cloud storage (AWS S3, Google Cloud Storage) for food images, receipts.

D. External Services

- **Maps & Location API:** Google Maps / Mapbox (for GPS tracking).
- **Payment Gateway:** Razorpay, PayPal, Stripe.
- **Messaging Services:** Twilio, Firebase (for OTPs, order updates).

3. Data Flow (Example: Customer Orders Food)

1. **Customer** selects food → Request sent to backend API.
2. **Backend (Order Service):** Validates order → Sends to restaurant.
3. **Restaurant:** Accepts → Updates status.
4. **Backend:** Assigns delivery partner → Notifies via app.
5. **Delivery Partner:** Accepts → GPS navigation to restaurant & customer.
6. **Payment Service:** Handles transaction securely.
7. **Database:** Stores order, payment, and delivery status.
8. **Customer App:** Shows live tracking until delivery.

4. High-Level Architecture Diagram (Textual)

20. Design a basic system architecture for a food delivery app.

Ans. =

```
    [ Customer App ]       [ Restaurant App ]      [ Delivery Partner App ]

         |                            |                          |

         -------------(API Gateway / Backend Services) ------------

                                 |

         ---------------------------------------------------------------------------

              |                   |                   |                      |

         User Service      Order Service   Payment Service   Notification Service

             |                 |                   |                        |

         [User DB]       [Orders DB]      [Payments DB]   [Cache/Queue System]

                                |

                [Maps API] [Payment Gateway] [SMS/Email Push]
```

21.Develop test cases for a simple calculator program.

Ans. = . **Functional Test Cases**

Test Case ID	**Input**	**Operation**	**Expected Output**
TC_01	2 + 3	Addition	5
TC_02	-5 + 8	Addition	3
TC_03	10 - 4	Subtraction	6
TC_04	-3 - (-6)	Subtraction	3
TC_05	7 × 8	Multiplication	56
TC_06	-5 × 4	Multiplication	-20
TC_07	20 ÷ 4	Division	5
TC_08	7 ÷ 2	Division	3.5

22. Document a real-world case where a software application required critical maintenance.

Ans. =

- **Case Study: Critical Software Maintenance – Microsoft Windows XP Patch (WannaCry Ransomware Attack)**

Background

- In **May 2017**, the world was hit by the **WannaCry ransomware attack**.
- The ransomware spread rapidly across networks, encrypting data and demanding Bitcoin payments.
- It exploited a vulnerability in the **Server Message Block (SMB) protocol** in Windows systems

Issue

- The affected systems were mostly running **older, unpatched versions of Microsoft Windows**, including **Windows XP**, which had already reached its **end of support** in 2014.
- Since XP was no longer officially maintained, many organizations had outdated systems without critical security updates.

Maintenance Action Taken

- **Microsoft released an emergency patch (MS17-010)** for supported Windows versions (Windows 7, Windows 8.1, Windows 10, and Windows Server editions).
- Due to the severity of the attack, Microsoft took the unusual step of releasing a **special security update for Windows XP, Windows Server 2003, and Windows 8**, even though they were officially out of support.
- IT administrators worldwide had to **urgently apply patches, disable SMBv1, and isolate infected systems** to prevent further spread.

Impact

- WannaCry infected more than **200,000 computers across 150 countries**.
- Critical organizations like the **UK's National Health Service (NHS)** were severely disrupted — hospitals couldn't access patient records, delaying treatments and surgeries.
- The total financial impact was estimated in the **billions of dollars**.

Lessons Learned

1. **Importance of Regular Updates & Patching**
   - Organizations must apply security patches promptly.
2. **Extended Maintenance for Critical Systems**
   - Even outdated software (like XP) may require emergency maintenance when still in use.
3. **Proactive Security Practices**
   - Use of firewalls, network segmentation, and disabling unused protocols can reduce risk.
4. **End-of-Life (EOL) Risk**
   - Continuing to use unsupported software increases vulnerability exposure.

23. Create a DFD for a hospital management system.

Ans. = Data Flow Diagram for Hospital Management System

Level 0 (Context Diagram)

Entities:

- **Patient** → Registers, books appointments, makes payments.
- **Doctor** → Provides diagnosis, updates medical records.
- **Admin/Receptionist** → Manages staff, patients, billing.
- **Pharmacy** → Dispenses medicines.
- **Laboratory** → Conducts tests and updates reports.

System: Hospital Management System (HMS)

Data Flows:

- Patient Info, Appointments, Prescriptions, Bills, Test Reports.

Textual Representation:

```
[Patient] ------------> (Hospital Management System) <------------ [Doctor]

      |                  |                     |

      |                  |                     |

      v                  v                     v

 [Pharmacy]             [Admin/Reception]          [Laboratory]
```

Level 1 DFD (Expanded View of HMS)

Main Processes:

1. **Patient Management**
   - Input: Patient registration details (Name, Age, Contact).
   - Output: Patient records stored in database.
2. **Appointment Scheduling**
   - Input: Patient appointment request.
   - Output: Doctor's schedule confirmation.
3. **Medical Records & Diagnosis**
   - Input: Doctor updates prescriptions & diagnosis.
   - Output: Stored in patient medical history.
4. **Billing & Payments**
   - Input: Services used (consultation, lab, pharmacy).
   - Output: Invoice, receipt, payment confirmation.
5. **Pharmacy Management**
   - Input: Prescription from doctor.
   - Output: Medicines dispensed to patient.
6. **Laboratory Management**
   - Input: Test request from doctor.
   - Output: Test reports stored & shared with doctor/patient.

Textual Flow Representation:

```
[Patient] ---> (1. Patient Management) ---> [Patient Database]



[Patient] ---> (2. Appointment Scheduling) ---> [Doctor Schedule DB] ---> [Doctor]



[Doctor] ---> (3. Medical Records & Diagnosis) ---> [Medical Records DB] ---> [Patient]
```

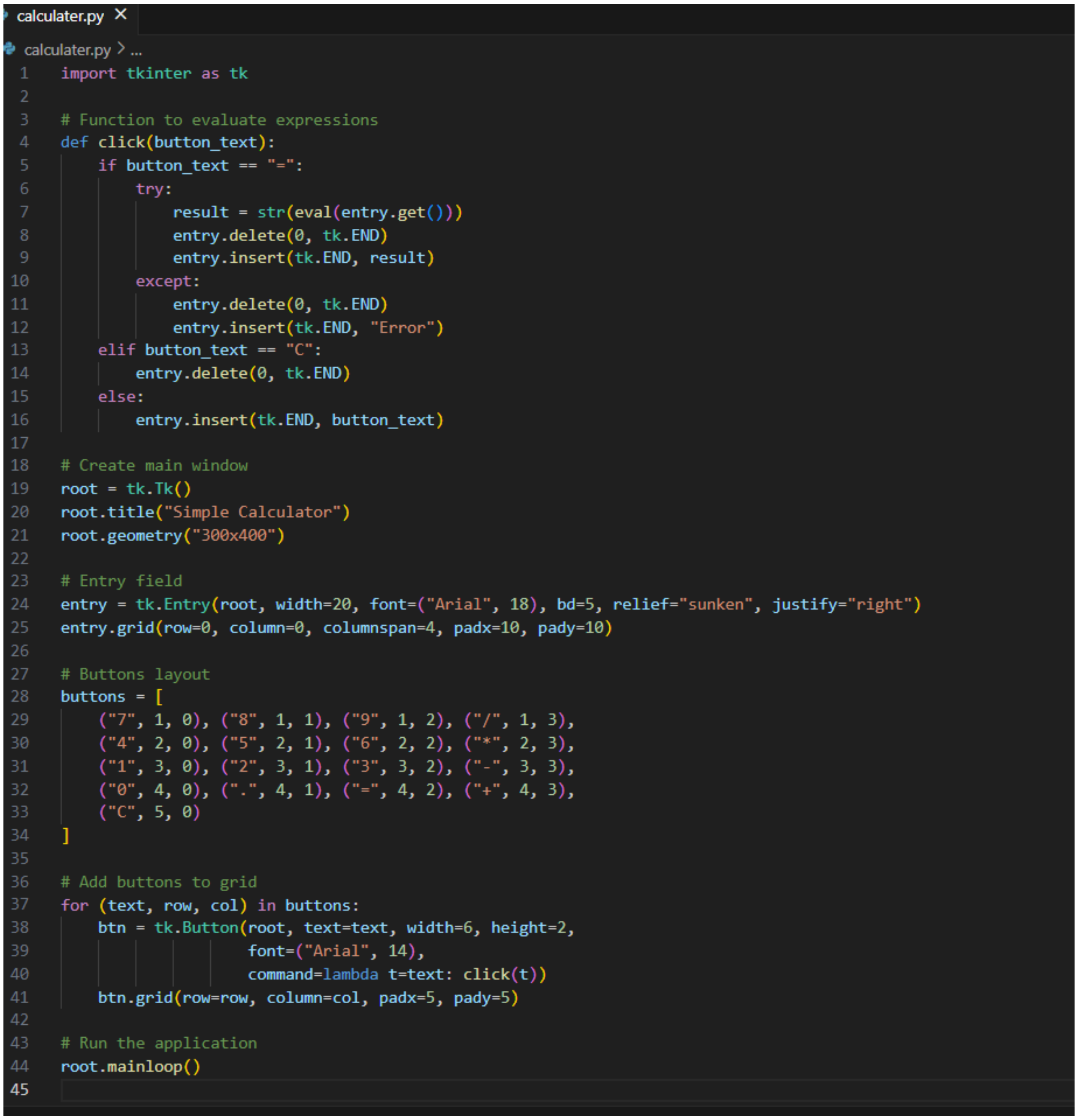
[Patient] ---> (4. Billing & Payments) ---> [Billing Database] ---> [Admin]

[Doctor] ---> (5. Pharmacy Management) ---> [Pharmacy Database] ---> [Pharmacy]

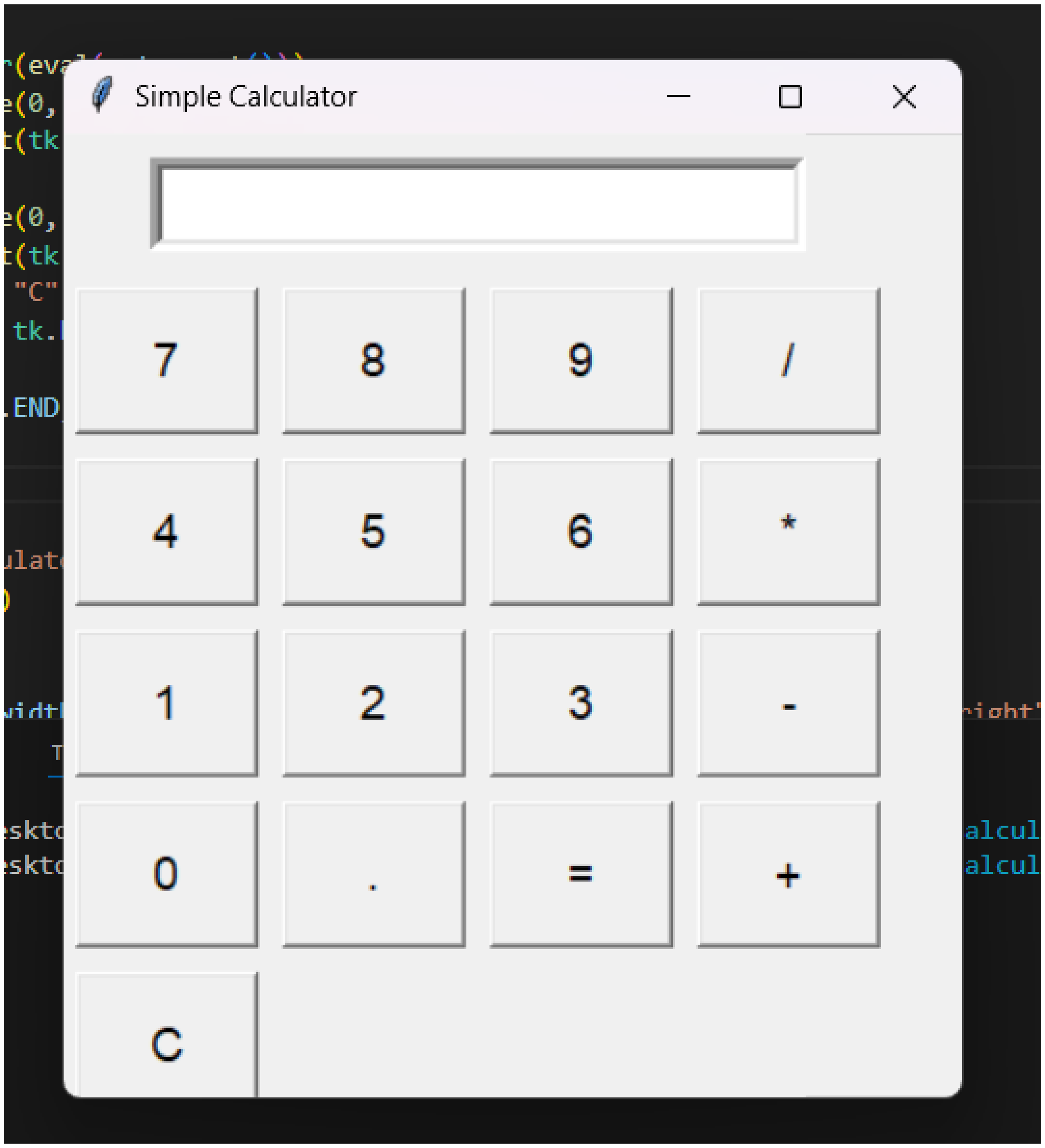
[Doctor] ---> (6. Laboratory Management) ---> [Lab Database] ---> [Laboratory]

24. Build a simple desktop calculator application using a GUI library.

Ans. =

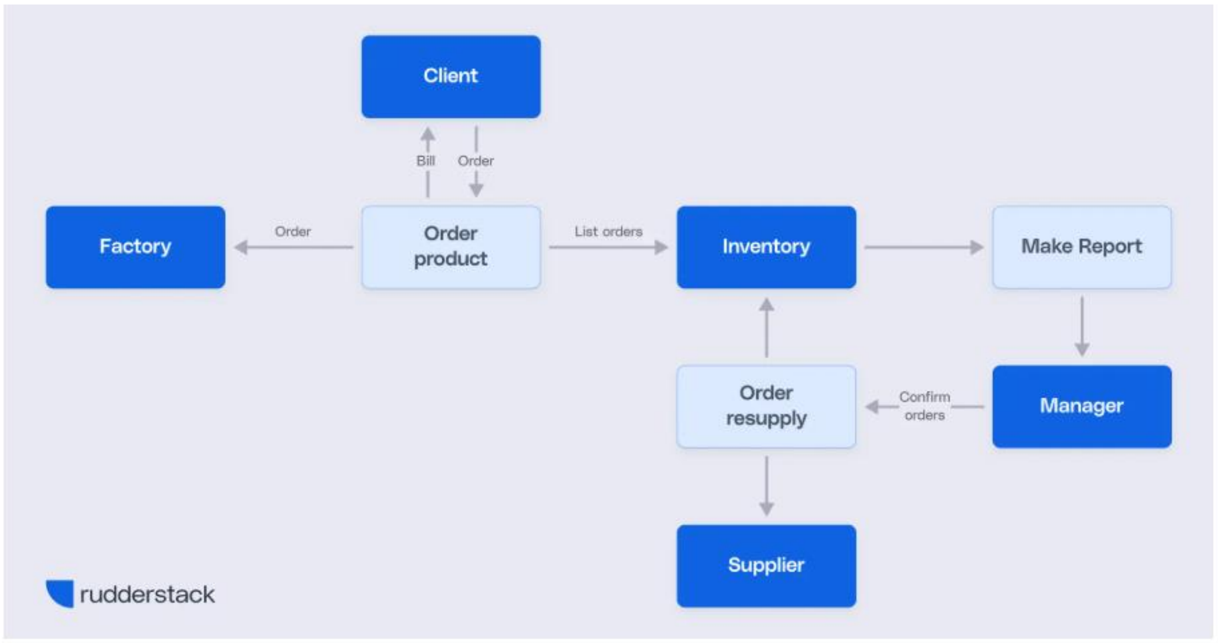


```python
import tkinter as tk

# Function to evaluate expressions
def click(button_text):
    if button_text == "=":
        try:
            result = str(eval(entry.get()))
            entry.delete(0, tk.END)
            entry.insert(tk.END, result)
        except:
            entry.delete(0, tk.END)
            entry.insert(tk.END, "Error")
    elif button_text == "C":
        entry.delete(0, tk.END)
    else:
        entry.insert(tk.END, button_text)

# Create main window
root = tk.Tk()
root.title("Simple Calculator")
root.geometry("300x400")

# Entry field
entry = tk.Entry(root, width=20, font=("Arial", 18), bd=5, relief="sunken", justify="right")
entry.grid(row=0, column=0, columnspan=4, padx=10, pady=10)

# Buttons layout
buttons = [
    ("7", 1, 0), ("8", 1, 1), ("9", 1, 2), ("/", 1, 3),
    ("4", 2, 0), ("5", 2, 1), ("6", 2, 2), ("*", 2, 3),
    ("1", 3, 0), ("2", 3, 1), ("3", 3, 2), ("-", 3, 3),
    ("0", 4, 0), (".", 4, 1), ("=", 4, 2), ("+", 4, 3),
    ("C", 5, 0)
]

# Add buttons to grid
for (text, row, col) in buttons:
    btn = tk.Button(root, text=text, width=6, height=2,
                    font=("Arial", 14),
                    command=lambda t=text: click(t))
    btn.grid(row=row, column=col, padx=5, pady=5)

# Run the application
root.mainloop()

```



25. Draw a flowchart representing the logic of a basic online registration system.

Ans. =


Client
Bill
Order
Factory
Order
Order product
List orders
Inventory
Make Report
Order resupply
Confirm orders
Manager
Supplier
rudderstack