**MODULE 1- SE- Overview of IT industry**

1) Write a simple "Hello World" program in two different programming languages

Of your choice. Compare the structure and syntax.

Ans = done. in the language python and c.

2) Research and create a diagram of how data is transmitted from a client to a

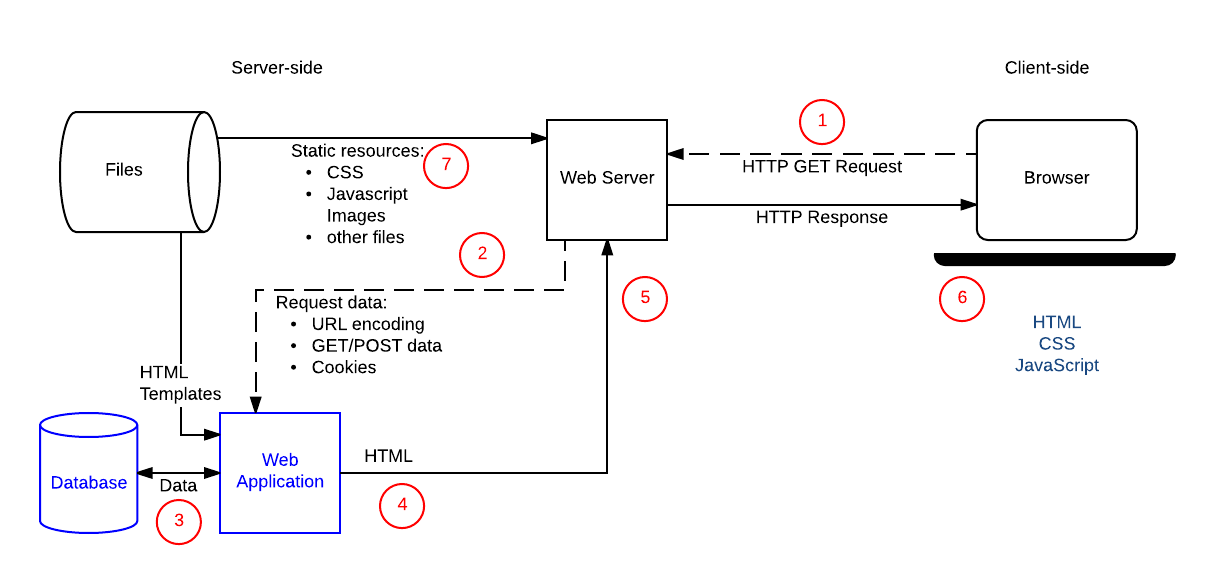
Server over the internet.

**How Data is Transmitted from a Client to a Server Over the Internet**

When a client (e.g., a web browser on a user's device) sends a request to a server (e.g., a website server), the following steps take place:

1. **Client Request**:
   * The client enters a URL in the browser, which creates an **HTTP request** for a specific resource (such as a webpage or image).
2. **DNS Lookup**:
   * The browser sends the domain name (e.g., www.example.com) to a **DNS server** to get the corresponding **IP address** of the server.
3. **TCP Connection**:
   * The client establishes a **TCP connection** with the server using the **IP address**. A **three-way handshake** is performed to ensure the connection is reliable.
4. **Data Transfer**:
   * The client sends the HTTP request to the server. The server processes the request, retrieves the resource (e.g., HTML, images), and sends the **HTTP response** back to the client.
5. **Client Displays Data**:
   * The client receives the server’s response and renders the data, displaying the webpage or content to the user.

**Diagram:**



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

Ans= this program done in python

10) Research different types of internet connections (e.g., broadband, fiber,

satellite) and list their pros and cons.

**Types of Internet Connections with Pros and Cons**

1. **Broadband (DSL/Cable)**
   * **Description**: High-speed internet delivered via telephone lines (DSL) or cable TV lines (Cable).
   * **Pros**:
     + Widely available
     + Always-on connection
     + Affordable plans
   * **Cons**:
     + Slower speeds compared to fiber
     + Speed may vary during peak hours
2. **Fiber-Optic Internet**
   * **Description**: Uses fiber-optic cables to transmit data at very high speeds using light signals.
   * **Pros**:
     + Extremely fast and reliable
     + High bandwidth for multiple users/devices
     + Low latency
   * **Cons**:
     + Expensive installation
     + Limited availability in rural areas
3. **Satellite Internet**
   * **Description**: Internet provided via satellites orbiting the Earth, ideal for remote or rural areas.
   * **Pros**:
     + Available almost anywhere
     + Good option for areas without cable/fiber
   * **Cons**:
     + High latency (signal delay)
     + Weather can affect signal quality
     + Data caps and slower speeds
4. **Mobile Data (4G/5G)**
   * **Description**: Internet access via mobile networks on smartphones or mobile hotspots.
   * **Pros**:
     + Portable and wireless
     + Widely available in urban areas
     + Fast speeds (especially with 5G)
   * **Cons**:
     + Data limits on many plans
     + Speed varies with signal strength and location
5. **Dial-Up** *(Outdated)*
   * **Description**: Uses telephone lines for internet; very slow. Rarely used today.
   * **Pros**:
     + Very low cost
     + Simple setup
   * **Cons**:
     + Extremely slow (56 Kbps max)
     + Ties up phone line
     + Not suitable for modern use

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

**🌐 1. Simulate an HTTP Request using curl**

**🔹 GET Request**

curl http://example.com

* This fetches the HTML content of the homepage at example.com.

**🔹 GET Request with headers**

curl -H "User-Agent: MyBrowser" http://example.com

**🔹 POST Request with data**

curl -X POST -d "username=test&password=1234" http://example.com/login

**🔹 Save output to file**

curl -o output.html http://example.com

**📁 2. Simulate an FTP Request using curl**

You can use curl to **download** or **upload** files via FTP.

**🔹 Download a file from an FTP server**

curl ftp://ftp.example.com/file.txt --user username:password

**🔹 Upload a file to an FTP server**

curl -T localfile.txt ftp://ftp.example.com/ --user username:password

14) Identify and explain three common application security vulnerabilities. Suggestpossible solutions.

**1. 🔐 SQL Injection (SQLi)**

**✅ What it is:**

A vulnerability that allows attackers to inject malicious SQL queries into input fields, potentially gaining access to or modifying database data.

**❌ Example:**

Sql

SELECT \* FROM users WHERE username = 'admin' AND password = '123' OR '1'='1';

**✅ Solution:**

* **Use Prepared Statements (Parameterized Queries)**  
  In Python (with SQLite):

python

CopyEdit

cursor.execute("SELECT \* FROM users WHERE username = ? AND password = ?", (user, pwd))

* **Input Validation**: Sanitize and validate all inputs.
* **Least Privilege**: Ensure the database user has minimal rights.

**2. 🧱 Cross-Site Scripting (XSS)**

**✅ What it is:**

An attacker injects malicious scripts into web pages viewed by other users. Often used to steal cookies, session tokens, or deface pages.

**❌ Example:**

<script>alert('Hacked!');</script>

**✅ Solution:**

* **Escape Output**: Encode HTML, JavaScript, and URL outputs to prevent script execution.
* **Content Security Policy (CSP)**: Prevent loading untrusted scripts.
* **Input Sanitization**: Remove or reject suspicious input content.

**3. 🔑 Broken Authentication**

**✅ What it is:**

Weak session management or poor password handling allows attackers to impersonate users.

**❌ Issues include:**

* Default credentials
* Insecure password storage
* Exposed session tokens

**✅ Solution:**

* **Use Secure Authentication Libraries** (e.g., OAuth, JWT)
* **Implement Multi-Factor Authentication (MFA)**
* **Hash Passwords** with strong algorithms like bcrypt
* **Secure Session Management**: Regenerate session IDs after login, use HTTPS.

**🧠 Final Tip**

To build secure applications:

* Follow the **OWASP Top 10** security risks
* Perform regular **code reviews and penetration testing**
* Keep libraries and frameworks **up to date**

16) Identify and classify 5 applications you use daily as either system software

orapplication software

 **Google Chrome** – This is an application software used for browsing the internet.

 **Microsoft Word** – An application software used for creating and editing documents.

 **Windows (or Linux)** – This is system software that manages hardware and runs other software.

 **File Explorer** – A part of system software used to manage files and folders on your device.

 **Spotify** – An application software used for streaming music and podcasts.

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



Sure! Here's a short explanation of a **three-tier architecture** for a web application:

1. **Presentation Tier**: The front-end (e.g., web browser or mobile app) that users interact with.
2. **Application Tier**: The back-end logic (e.g., web server or app server) that processes requests and handles business logic.
3. **Data Tier**: The database server where data is stored, retrieved, and managed.

20) Create a case study on the functionality of the presentation, business logic, and

dataaccess layers of a given software system.

### 🧾 ****Case Study: Online Bookstore System****

#### 1. ****Presentation Layer (User Interface)****

* **Function:** Handles user interaction.
* **Example:** A web interface where users can browse books, search by title, add items to a cart, and check out.
* **Technology Used:** HTML, CSS, JavaScript, React

#### 2. ****Business Logic Layer (Application Logic)****

* **Function:** Processes rules and workflows.
* **Example:** Applies discount rules, validates payment info, calculates totals, manages user sessions.
* **Technology Used:** Node.js, Java, or Python (e.g., Flask/Django)

#### 3. ****Data Access Layer****

* **Function:** Connects to the database, handles queries.
* **Example:** Fetches book data, saves orders, updates inventory.
* **Technology Used:** SQL queries via ORM (e.g., SQLAlchemy, Hibernate), direct DB APIs

### 🎯 Summary:

* The **Presentation layer** is what users see and use.
* The **Business logic layer** is the brain of the system.
* The **Data access layer** is responsible for talking to the database.

22) Explore different types of software environments (development, testing,

Production ).Set up a basic environment in a virtual machine.

**🧪 Types of Software Environments**

1. **Development Environment**
   * Used by developers to write and debug code.
   * May include IDEs, compilers, and local servers.
   * Example: Local setup with Visual Studio Code and XAMPP.
2. **Testing Environment**
   * Used to test features and bug fixes.
   * Mimics production, but with test data.
   * Tools: Selenium, Postman, JUnit, etc.
3. **Production Environment**
   * Live environment used by end-users.
   * Requires high performance, security, and stability.
   * Example: A deployed web app on AWS or a live server.

**💻 Set Up a Basic Environment in a Virtual Machine (VM)**

**Step-by-step using VirtualBox + Ubuntu VM:**

1. **Install VirtualBox**  
   Download from <https://www.virtualbox.org>
2. **Download Ubuntu ISO**  
   Get from <https://ubuntu.com/download/desktop>
3. **Create a New VM**
   * Allocate memory (e.g., 2GB RAM)
   * Attach Ubuntu ISO as bootable disk
4. **Install Ubuntu on the VM**  
   Follow prompts to install the OS.
5. **Set Up a Development Environment**  
   In the Ubuntu VM:

sudo apt update

sudo apt install git python3-pip nodejs npm

1. **Optional: Set up Testing Tools**  
   Install tools like pytest, Postman, or Selenium.

24) Write and upload your first source code file to Github.

* # hello.py

print("Hello, GitHub!")

* # Initialize local repo (in your project folder)

git init

git add hello.py

git commit -m "First commit"

# Connect to your GitHub repo

git remote add origin https://github.com/your-username/my-first-code.git

git branch -M main

git push -u origin main

26)Create a Github repository and document how to commit and push code changes

28) Create a student account on Github and collaborate on a small project

with a classmate.

30) Create a list of software you use regularly and classify them into the

Following categories: system, application, and utility software.

Sure! Here's a simple list of software I (or most people) use regularly, classified into \*\*System\*\*, \*\*Application\*\*, and \*\*Utility\*\* software:

---

### 🖥️ \*\*System Software\*\*

- \*\*Windows / macOS / Linux\*\* – Operating Systems that manage hardware and run applications.

- \*\*Device Drivers\*\* – Enable communication between hardware (like printers, keyboards) and OS.

---

### 📱 \*\*Application Software\*\*

- \*\*Google Chrome / Firefox\*\* – Web browsers for accessing the internet.

- \*\*Microsoft Word / Excel\*\* – Office tools for writing and data analysis.

- \*\*Spotify\*\* – For streaming music.

- \*\*Zoom / Teams\*\* – For video conferencing.

- \*\*WhatsApp / Telegram (Desktop)\*\* – For messaging and collaboration.

---

### 🛠️ \*\*Utility Software\*\*

- \*\*Antivirus (e.g., Windows Defender, Avast)\*\* – Protects against malware.

- \*\*WinRAR / 7-Zip\*\* – File compression tools.

- \*\*CCleaner\*\* – Cleans up junk files and optimizes system.

- \*\*Backup Software (e.g., Time Machine, OneDrive Backup)\*\* – For data backup and recovery.

- \*\*Disk Management Tools\*\* – Manage partitions and storage space.

---

32) Follow a GIT tutorial to practice cloning, branching, and merging repositories

Great! Here’s a simple step-by-step \*\*Git tutorial\*\* to help you \*\*practice cloning, branching, and merging\*\* a repository. You can try this on your own GitHub repo or a sample one.

---

## 🔁 Step 1: \*\*Clone a Repository\*\*

1. Create or find a GitHub repo (e.g., `https://github.com/your-username/git-practice`)

2. Clone it to your computer:

```bash

git clone https://github.com/your-username/git-practice.git

cd git-practice

```

---

## 🌿 Step 2: \*\*Create and Switch to a New Branch\*\*

```bash

git checkout -b feature-1

```

This creates a new branch called `feature-1` and switches to it.

---

## 📝 Step 3: \*\*Make Changes and Commit\*\*

1. Create or edit a file (e.g., `hello.txt`)

2. Add content:

```bash

echo "Hello from feature-1" > hello.txt

```

3. Stage and commit your changes:

```bash

git add hello.txt

git commit -m "Added hello.txt in feature-1 branch"

```

## 🔄 Step 4: \*\*Switch Back to Main and Merge\*\*

```bash

git checkout main

git merge feature-1

```

This merges changes from `feature-1` into `main`.

---

## 🧹 Optional: \*\*Delete the Branch (Cleanup)\*\*

```bash

git branch -d feature-1

```

---

## ⬆️ Step 5: \*\*Push Changes to GitHub\*\*

```bash

git push origin main

```

If pushing a new branch:

```bash

git push origin feature-1

```

34) Write a report on the various types of application software and how they

Improve productivity.

**📄 Report: Types of Application Software and Their Role in Productivity**

**1. Word Processing Software**

* **Examples:** Microsoft Word, Google Docs
* **Use:** Creating, editing, and formatting text documents.
* **Productivity Boost:** Enables fast and professional document creation with tools like spell-check, templates, and collaboration.

**2. Spreadsheet Software**

* **Examples:** Microsoft Excel, Google Sheets
* **Use:** Managing data, performing calculations, creating charts.
* **Productivity Boost:** Automates calculations, visualizes data, and supports financial planning or analysis.

**3. Presentation Software**

* **Examples:** Microsoft PowerPoint, Google Slides
* **Use:** Designing slides for lectures, meetings, and proposals.
* **Productivity Boost:** Helps communicate ideas clearly with visuals, animations, and templates.

**4. Database Management Software**

* **Examples:** Microsoft Access, MySQL
* **Use:** Storing, organizing, and retrieving structured data.
* **Productivity Boost:** Speeds up data handling and supports accurate, quick decision-making.

**5. Communication Software**

* **Examples:** Zoom, Microsoft Teams, Slack
* **Use:** Video calls, messaging, and collaboration.
* **Productivity Boost:** Enhances teamwork and remote communication, saving time and travel.

**6. Graphic Design Software**

* **Examples:** Adobe Photoshop, Canva
* **Use:** Creating visual content like posters, ads, or UI designs.
* **Productivity Boost:** Helps professionals and marketers quickly create appealing visuals.

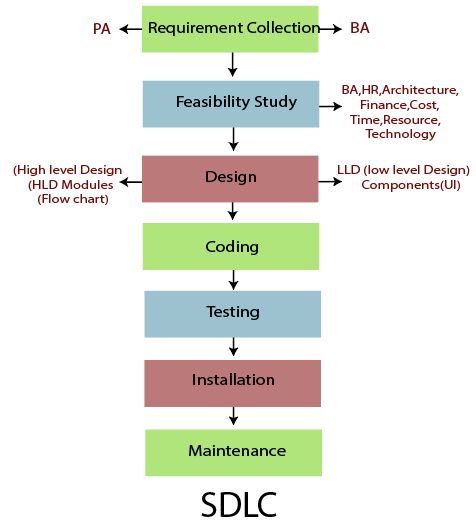
**7. Web Browsers**

* **Examples:** Google Chrome, Mozilla Firefox
* **Use:** Accessing web content and cloud applications.
* **Productivity Boost:** Allows fast research, online collaboration, and access to web tools.

**✅ Conclusion**

Application software plays a vital role in simplifying tasks, automating repetitive work, and improving accuracy. Whether for business, education, or creative work, these tools help users work **faster**, **smarter**, and **more efficiently**.

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



38) Write a requirement specification for a simple library management system.

**📄 Software Requirement Specification (SRS)**

**Project Name: Simple Library Management System**

**1. Introduction**

The Library Management System (LMS) is a software application designed to automate and manage library operations such as adding books, issuing books, returning books, and tracking users.

**2. Functional Requirements**

1. **User Authentication**
   * Admin and Librarian login with username and password.
2. **Book Management**
   * Add, update, delete, and search books.
   * Assign unique Book IDs.
3. **Member Management**
   * Register new members (students/faculty).
   * Edit or delete member details.
4. **Issue & Return System**
   * Issue books to members (with due dates).
   * Accept returned books and update availability.
5. **Search Function**
   * Search books by title, author, or category.
   * Search members by name or ID.
6. **Reports**
   * View reports on issued books, returned books, overdue books.

**3. Non-Functional Requirements**

* **Usability**: Simple and user-friendly interface.
* **Performance**: Fast response for book/member searches.
* **Scalability**: Should support multiple users and records.
* **Security**: Password-protected access, limited privileges.

**4. System Requirements**

* **Platform**: Web-based or Desktop
* **Frontend**: HTML/CSS/JavaScript or GUI Framework
* **Backend**: Python, Java, or PHP
* **Database**: MySQL / SQLite / PostgreSQL

**5. Users of the System**

* **Librarian/Admin**: Full access to all functionalities.
* **Members (Optional)**: Can view and request books online (in extended version).

40) Perform a functional analysis for an online shopping system.

**🛒 Functional Analysis: Online Shopping System**

**✅ 1. User Registration & Login**

* **Function:** Allows customers to create accounts and log in securely.
* **Inputs:** Username, password, email.
* **Outputs:** Account creation confirmation or login success/error.

**🛍️ 2. Product Catalog**

* **Function:** Displays all products with images, descriptions, and prices.
* **Inputs:** Search terms, filters (e.g., category, price range).
* **Outputs:** List of matching products.

**➕ 3. Shopping Cart**

* **Function:** Lets users add, remove, or update items.
* **Inputs:** Product selections and quantities.
* **Outputs:** Updated cart with total price.

**💳 4. Checkout and Payment**

* **Function:** Handles order placement and payment processing.
* **Inputs:** Shipping address, payment details.
* **Outputs:** Order confirmation, invoice, and email receipt.

**🚚 5. Order Management**

* **Function:** Tracks order status (e.g., pending, shipped, delivered).
* **Inputs:** Order ID or user account.
* **Outputs:** Status updates and delivery info.

**👤 6. User Profile**

* **Function:** Allows users to view and edit personal info.
* **Inputs:** New user details (address, phone).
* **Outputs:** Confirmation of updates.

**💬 7. Customer Feedback/Reviews**

* **Function:** Users can leave reviews and rate products.
* **Inputs:** Text review, star rating.
* **Outputs:** Displayed on product page.

42) Perform a functional analysis for an online shopping system.

Sure! Here's a clear and short \*\*functional analysis\*\* for an \*\*Online Shopping System\*\*:

🛒 \*\*Functional Analysis: Online Shopping System\*\*

1. \*\*User Account Management\*\*

- \*\*Function:\*\* Users can register, log in, and manage their profiles.

2. \*\*Product Browsing & Search\*\*

- \*\*Function:\*\* Users can view product categories, search by name, filter by price, brand, or ratings.

3. \*\*Shopping Cart\*\*

- \*\*Function:\*\* Add, update, or remove products before purchasing.

4. \*\*Order Placement\*\*

- \*\*Function:\*\* Checkout with selected items, enter shipping info, choose payment method.

5. \*\*Payment Processing\*\*

- \*\*Function:\*\* Securely handle transactions via credit card, UPI, or other gateways.

6. \*\*Order Tracking\*\*

- \*\*Function:\*\* Users can view order history and track delivery status.

7. \*\*Reviews & Ratings\*\*

- \*\*Function:\*\* Users can leave feedback on purchased products.

8. \*\*Admin Functions\*\*

- \*\*Function:\*\* Admins manage product listings, view orders, and monitor users.

44) Design a basic system architecture for a food delivery app.

Here's a basic system architecture for a food delivery app, broken down into its main components and how they interact:

\*\*1. Client Applications\*\*

These are the user-facing interfaces:

- \*\*Customer App\*\* (iOS, Android, Web)

- \*\*Restaurant App\*\* (Tablet/Web for order management)

- \*\*Delivery Driver App\*\* (Mobile for navigation and order updates)

- \*\*Admin Panel\*\* (Web for platform management)

\*\*2. Backend Services (API Layer)\*\*

Handles business logic and communication between client apps and the database.

- \*\*User Service\*\* (auth, profiles, roles)

- \*\*Order Service\*\* (cart, order placement, order tracking)

- \*\*Restaurant Service\*\* (menus, availability, ratings)

- \*\*Delivery Service\*\* (driver assignment, status updates)

- \*\*Notification Service\*\* (push, email, SMS)

- \*\*Payment Service\*\* (integration with Stripe, PayPal, etc.)

- \*\*Search & Discovery Service\*\* (restaurants, dishes, filters)

\*\*3. Database Layer\*\*

Handles persistent data storage.

- \*\*Relational DB (e.g., PostgreSQL/MySQL)\*\* for structured data (users, orders, menus)

- \*\*NoSQL DB (e.g., MongoDB, Redis)\*\* for session management, caching, real-time updates

\*\*4. Real-Time Services\*\*

Handles instant updates and live tracking.

- \*\*WebSocket Server\*\* or \*\*Firebase\*\* for:

- Live order status

- Real-time delivery tracking

- Chat between customer, driver, and restaurant

\*\*5. External Integrations\*\*

- \*\*Payment Gateways\*\* (Stripe, PayPal)

- \*\*Maps & Navigation\*\* (Google Maps, Mapbox)

- \*\*SMS/Email Services\*\* (Twilio, SendGrid)

- \*\*Authentication\*\* (OAuth, social login)

\*\*6. DevOps & Infrastructure\*\*

- \*\*Cloud Provider\*\* (AWS, GCP, Azure)

- \*\*API Gateway\*\* for routing and rate limiting

- \*\*Containerization\*\* with Docker and orchestration via Kubernetes

- \*\*CI/CD Pipeline\*\* for automated deployments

- \*\*Monitoring & Logging\*\* (Prometheus, Grafana, ELK Stack)

\*\*7. Security & Compliance\*\*

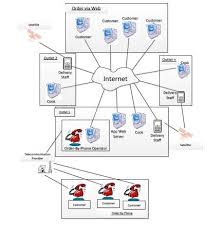
- SSL/TLS for secure communication

- Token-based authentication (JWT)

- Role-based access control (RBAC)

- Data encryption at rest and in transit

- GDPR and PCI compliance



46) Develop test cases for a simple calculator program.

Absolutely! Here’s a clean, non-tabular version of the test cases for a simple calculator program:

---

### \*\*Addition Test Cases\*\*

1. \*\*2 + 3 → 5\*\*

Standard addition of two positive integers.

2. \*\*-2 + 3 → 1\*\*

Adding a negative and a positive number.

3. \*\*-2 + (-3) → -5\*\*

Addition of two negative integers.

4. \*\*0 + 0 → 0\*\*

Adding two zeros.

5. \*\*1.5 + 2.3 → 3.8\*\*

Adding two floating-point numbers.

---

### \*\*Subtraction Test Cases\*\*

1. \*\*5 - 2 → 3\*\*

Regular subtraction resulting in a positive number.

2. \*\*2 - 5 → -3\*\*

Subtraction resulting in a negative number.

3. \*\*-3 - (-2) → -1\*\*

Subtracting one negative number from another.

4. \*\*0 - 0 → 0\*\*

Subtracting zero from zero.

5. \*\*5.5 - 2.2 → 3.3\*\*

Floating-point subtraction.

---

### \*\*Multiplication Test Cases\*\*

1. \*\*3 \* 4 → 12\*\*

Multiplication of two positive integers.

2. \*\*-3 \* 4 → -12\*\*

Negative multiplied by positive.

3. \*\*-2 \* -3 → 6\*\*

Negative multiplied by negative.

4. \*\*0 \* 5 → 0\*\*

Any number multiplied by zero should be zero.

5. \*\*1.5 \* 2 → 3.0\*\*

Floating-point multiplication.

---

### \*\*Division Test Cases\*\*

1. \*\*6 / 2 → 3.0\*\*

Standard division.

2. \*\*7 / 2 → 3.5\*\*

Division resulting in a float.

3. \*\*-6 / 2 → -3.0\*\*

Negative numerator.

4. \*\*-6 / -2 → 3.0\*\*

Both numerator and denominator negative.

5. \*\*0 / 5 → 0.0\*\*

Zero divided by any number.

6. \*\*5 / 0 → Error or Exception\*\*

Division by zero should be handled properly.

\*\*Edge Cases & Error Handling\*\*

1. \*\*"a" + 5 → Error or Exception\*\*

Invalid input (non-numeric).

2. \*\*Empty input → Error or Prompt\*\*

Should handle the case where no input is provided.

3. \*\*Very large numbers → Valid result or overflow handling\*\*

Checks system's ability to handle large values.

4. \*\*Special characters like "@" → Error or Exception\*\*

Non-numeric input should be validated and rejected.

48) Document a real-world case where a software application required

Critical maintenance.

**📌 Case Study: GitHub Outage (February 2020)**

**Background**

GitHub, the world’s largest code hosting platform, experienced a **major service outage** on February 27, 2020, that affected millions of developers globally. During the outage, users could not access repositories, perform Git operations, or interact with pull requests and issues.

**The Problem**

The outage was triggered by a **database cluster failure** in one of GitHub’s primary data centers. The root cause was traced to:

* A **corrupted database cluster member**, which led to replication lag and inconsistencies.
* Failure in automatic failover and recovery mechanisms.
* The load balancer continued routing traffic to the problematic node, worsening the situation.

**Impact**

* Services were partially or completely unavailable for **nearly 2 hours**.
* Global developer productivity was affected, especially teams reliant on CI/CD workflows tied to GitHub.
* Trust and reliability concerns emerged due to GitHub’s critical role in open-source and enterprise development.

**Critical Maintenance Actions**

GitHub's engineering team initiated an emergency maintenance operation which included:

1. **Taking the faulty node offline** and reconfiguring the cluster to restore replication integrity.
2. **Manually rerouting traffic** to healthy nodes.
3. **Scaling up read replicas** to absorb delayed requests.
4. Post-recovery, they applied:
   * **Automated failover improvements**.
   * **Additional monitoring for early anomaly detection**.
   * **System-wide redundancy testing** to prevent similar issues.

**Postmortem & Lessons Learned**

* GitHub publicly published a detailed [post-incident report](https://github.blog) to promote transparency.
* The incident underscored the importance of:
  + Robust **failover and disaster recovery planning**.
  + Continuous testing of **critical infrastructure under stress**.
  + Maintaining clear **incident response protocols**.

**Conclusion**

This case highlights how even the most sophisticated platforms are vulnerable to systemic failures, and how **critical maintenance** is not just about fixing the immediate problem, but also reinforcing the system to prevent future occurrences.

49) Create a DFD for a hospital management system

### 🏥 ****DFD – Level 0 (Context Level)****

#### ****Entities:****

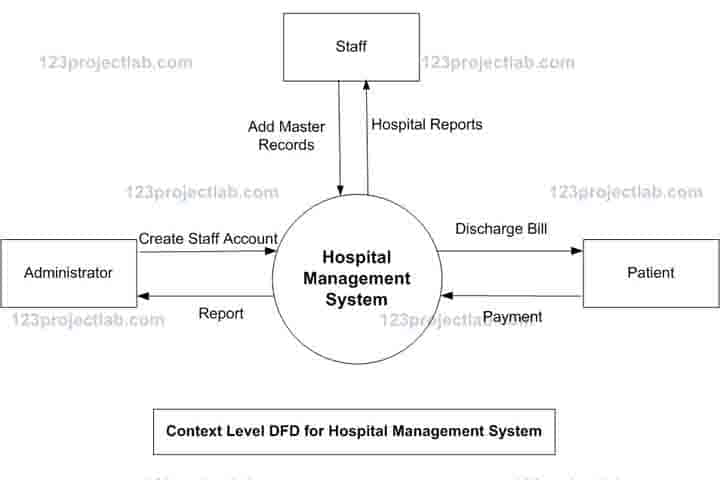
1. **Patient**
2. **Doctor**
3. **Admin/Staff**

#### ****Process:****

* **Hospital Management System**

#### ****Data Flows:****

* Patient → Registration Details → System
* System → Appointment Info → Patient
* Doctor → Availability Info → System
* System → Schedule & Patient Records → Doctor
* Admin → Staff/Data Management Info → System
* System → Reports/Billing → Admin



51) Build a simple desktop calculator application using a GUI library

Ans = done in python

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

