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

**1. Python**

Hello World in Python

print("Hello, World!")

**Explanation:**

* print() is a built-in function to display output.
* No need to define a class or function to run this.
* Very short and simple syntax.

**2. Java**

Hello World in Java

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

**Explanation:**

* Java is a **statically typed, object-oriented language**, so everything must be inside a **class**.
* The program starts execution from the main method: public static void main(String[] args)
* System.out.println() is used to print output to the console.

**Comparison:**

| **Feature** | **Python** | **Java** |
| --- | --- | --- |
| Simplicity | Very simple, single line | Requires boilerplate code |
| Structure | No class or main method needed | Must have a class and main method |
| Syntax | Minimal syntax | Verbose syntax |
| Execution | Interpreted | Compiled and then run using JVM |
| Typing | Dynamically typed | Statically typed |

**Summary:**

* **Python** is beginner-friendly and concise.
* **Java** is more structured and suitable for large applications, but requires more code for simple tasks.

**Research and create a diagram of how data is transmitted from a client to a serverover the internet.**

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

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

**Types of Internet Connections**

**1. Broadband (DSL/Cable)**

**Description**: Uses telephone (DSL) or TV cable lines (cable) to provide internet.

**Pros:**

* Widely available
* Always on (no dial-up needed)
* Affordable plans
* Suitable for general browsing and streaming

**Cons:**

* Slower than fiber
* Speed may drop during peak hours
* Limited upload speed

**2. Fiber-Optic**

**Description**: Uses light signals through fiber-optic cables to provide ultra-fast internet.

**Pros:**

* Super high speed (up to 1 Gbps or more)
* Excellent for streaming, gaming, and large downloads
* Reliable and stable connection
* Symmetrical upload/download speeds

**Cons:**

* More expensive
* Not available in all areas
* Installation may take time

**3. Satellite**

**Description**: Uses satellites to provide internet, usually in remote or rural areas.

**Pros:**

* Available almost anywhere (good for rural areas)
* Easy to set up where cables can't reach

**Cons:**

* Slower speeds and higher latency
* Weather can affect signal
* Expensive data plans
* Not great for gaming or video calls

**4. Mobile Data (4G/5G)**

**Description**: Uses mobile networks (like your phone) to connect to the internet.

**Pros:**

* Wireless and portable
* 5G offers high speed (comparable to broadband)
* Good for on-the-go usage

**Cons:**

* Data limits may apply
* Speed depends on signal strength
* Can be costly if overused

**5. Dial-up (Older technology)**

**Description**: Connects via telephone line and modem.

**Pros:**

* Very cheap
* Still used in some extremely remote areas

**Cons:**

* Very slow (maximum 56 kbps)
* Can’t use phone and internet at the same time
* Outdated and mostly obsolete

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

**1. Simulate HTTP Request (Website Access)**

**Purpose:**

To access a webpage or API using HTTP (like how a browser does it).

**Command:**

curl https://example.com

**What It Does:**

* Sends a request to the website example.com
* Shows the raw HTML content of the webpage in your terminal

**2. Simulate FTP Request (File Transfer)**

**Purpose:**

To download or upload files using the **FTP** protocol.

**Example: Downloading a file from FTP**

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

**What It Does:**

* Connects to the FTP server
* Logs in with a username and password
* Downloads file.txt from the server

**Example: Uploading a file to FTP**

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

**What It Does:**

* Sends your file myfile.txt to the FTP server

**Extra Tip: View HTTP Headers Only**

If you just want to **see the headers** (without the page content):

curl -I https://example.com

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

**1. SQL Injection – Tricking the Login Box**

**What is it?**

A hacker types something **weird** in a login form to break into the system without knowing the real password.

**Simple Example:**

Imagine a school gate that asks for your name.  
Instead of giving a real name, someone says:

"Let everyone in!"

The guard (system) gets confused and lets the person in

**Simple Solution:**

* Don’t just trust whatever the user types.
* Make sure the system checks it carefully.

**2. Cross-Site Scripting (XSS) – Bad Messages on Website**

**What is it?**

Someone puts a **bad script** in a comment or message on your website that can pop up messages or steal data.

**Simple Example:**

Imagine this:

You're on a recipe-sharing website. Someone posts a comment on your cupcake recipe that says:

“Nice recipe!”  
(But behind the scenes, they’ve added some sneaky code.)

Now, when **anyone** reads that comment, a pop-up appears saying:

“You’ve been logged out – log in again.”

If someone logs in again, their **username and password** go straight to the attacker.

**Simple Solution:**

* Don’t allow strange symbols in comments (like < or >).
* Clean the message before showing it.

**3. Broken Authentication – Weak Login**

**What is it?**

The login system is too **easy to break**, like no strong passwords or unlimited tries.

**Simple Example:**

* Your password is just 1234.
* Anyone can try 1234, 0000, abcd until they get in. No one stops them.

**Simple Solution:**

* Use strong passwords like Cool@123.
* Block login after 3-5 wrong tries.
* Add an extra layer like OTP or code on phone (2-step login).

**Quick Summary (No Coding, Just Logic)**

| **Problem** | **In Simple Words** | **Fix in Simple Words** |
| --- | --- | --- |
| **SQL Injection** | Trick the login to break in | Check what people type |
| **XSS** | Put bad message that affects others | Don’t allow special code in comments |
| **Weak Login System** | Password too easy, no protection | Strong password + limit login attempts |

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

1. **Google Chrome** – Application Software
2. **WhatsApp** – Application Software
3. **Windows 10** – System Software
4. **MS Word** – Application Software
5. **Android OS** – System Software

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

* **Presentation Tier:** The user interface where the end-user interacts with the system (e.g., a web browser or a mobile app).
* **Logic Tier:** The middle tier of the architecture, also known as the logic tier, handles the application’s core processing, business rules, and calculations.
* **Data Tier:** Manages the storage, retrieval, and manipulation of the application’s data, typically utilizing a database.

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

A **case study** on a simple **Online Food Delivery System** (like Zomato or Swiggy) that explains the functionality of the **Presentation Layer**, **Business Logic Layer**, and **Data Access Layer** in **simple terms**.

**Case Study: Online Food Delivery System**

**Software System Goal:**

Allow users to browse restaurants, order food, and track delivery using a mobile or web app.

**1. Presentation Layer (Frontend/UI Layer)**

**Purpose:**

To interact with the user and show them information like restaurant lists, menu items, cart, and order status.

**Example Functions:**

* Display list of nearby restaurants
* Show food menu with images and prices
* Let users add food to cart
* Show order status (e.g., "Preparing", "Out for delivery")

**Simple Example:**

When a user taps "Order Now" on a burger, this button belongs to the presentation layer.

**2. Business Logic Layer (Middle Layer)**

**Purpose:**

Handles **all the rules** and **decision-making** behind the scenes.

**Example Functions:**

* Check if the restaurant is open or closed
* Calculate total price, including tax and delivery fee
* Apply promo codes
* Verify if delivery address is serviceable
* Send confirmation to the user and the restaurant

**Simple Example:**

When a promo code is applied, this layer checks if it's valid and calculates the discount.

**3. Data Access Layer (Backend/Database Layer)**

**Purpose:**

Talks to the **database**. It **stores** and **retrieves** data like user info, orders, menu items, etc.

**Example Functions:**

* Save a new food order in the database
* Fetch restaurant list from the database
* Update order status as "Delivered"
* Store customer feedback and ratings

**Simple Example:**

When you check your order history, this layer fetches all past orders from the database.

**How All Layers Work Together:**

* User -> clicks "Order Pizza" → Presentation Layer sends request → Business Layer checks restaurant status, applies offers → Data Layer saves order to database →Business Layer confirms →Presentation Layer shows "Order Placed Successfully"

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

**What Are Software Environments?**

Software environments are **places or stages** where software is run or tested during development. These help developers make sure everything works properly before users use the final product.

**Types of Software Environments:**

**1. Development Environment**

* **Used by:** Programmers
* **Purpose:** Writing and building the software
* **Example:** A developer writes code on their own computer using tools like VS Code, Java, Python, etc.
* **Think of it like:** A kitchen where the cook is still preparing the dish.

**2. Testing Environment**

* **Used by:** Testers or developers
* **Purpose:** To find bugs before software goes to real users
* **Example:** The software is tested with sample data to see if it crashes or misbehaves.
* **Think of it like:** A food taster checks the meal before it goes to the customer.

**3. Production Environment**

* **Used by:** Real users
* **Purpose:** The final version that people use
* **Example:** The actual website or app that you open in your browser or phone.
* **Think of it like:** The final dish being served to the customer.

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

**1. Create a GitHub Account**

* Go to [https://github.com](https://github.com/)
* Click **Sign up** and follow the steps.

**2. Create a New Repository**

1. After logging in, click the **"+" icon** on the top right → choose **"New repository"**
2. Fill in the details:
   * **Repository name**: e.g., hello-world
   * Description (optional)
   * Choose **Public** or **Private**
   * Tick **"Add a README file"**
3. Click **Create repository**

**3. Create a Simple Source Code File**

Write a simple program on your computer (example: in Notepad or any code editor):

Example: hello.py (Python file)

print("Hello, GitHub!")

Save it on your desktop or any folder.

**4. Upload Code File to GitHub**

1. Go to your repository on GitHub.
2. Click the **“Add file”** button → Choose **“Upload files”**
3. Click **“Choose your files”** → Select your hello.py file from your computer.
4. Scroll down and click **"Commit changes"**

**Create a GitHub repository and document how to commit and push code changes**

**Step 1: Install GitHub Desktop**

1. Go to [https://desktop.github.com](https://desktop.github.com/)
2. Click **Download for Windows/Mac**
3. Install it (just like any normal app)

**Step 2: Sign In to GitHub**

1. Open GitHub Desktop
2. Click **"Sign in to GitHub.com"**
3. Log in with your GitHub username and password

**Step 3: Create a Repository (Online)**

You need an online "project folder" first.

1. Go to [https://github.com](https://github.com/)
2. Click **+** (top-right corner) → **New repository**
3. Fill in:
   * Name: my-first-project
   * Check "Add a README"
4. Click **Create repository**

**Step 4: Clone Repository to Your Computer**

Now let’s bring your GitHub repo to your PC:

1. Open GitHub Desktop
2. Go to **File > Clone repository**
3. Go to the **"GitHub.com" tab**
4. Find your repo (my-first-project)
5. Choose a folder location (where to save it)
6. Click **Clone**

Now it’s saved on your computer!

**Step 5: Add or Edit a File**

1. Open the folder on your PC (e.g., my-first-project)
2. Create a new file (e.g., hello.txt)
3. Write something like:
4. Hello from GitHub Desktop!
5. Save it

**Step 6: Commit and Push to GitHub (Upload)**

1. Go back to **GitHub Desktop**
2. It will show the file you added (hello.txt)
3. At the bottom, write a **summary**:
   * e.g., Added hello.txt file
4. Click the blue **Commit to main** button
5. Then click **Push origin** (top bar)

| **Action** | **What It Means** |
| --- | --- |
| **Clone** | Downloaded project from GitHub |
| **Commit** | Saved your changes |
| **Push** | Uploaded changes to GitHub |

**Create a student account on GitHub and collaborate on a small project with a classmate.**

**Step 1: Create a Student GitHub Account**

1. Go to [https://github.com](https://github.com/)
2. Click **"Sign up"** (top-right corner)
3. Fill in your:
   * Username
   * Email
   * Password
4. Verify your email
5. Done! You now have a GitHub account

**Step 2: Create a New Project (Repository)**

1. Go to your GitHub profile
2. Click **+ → New repository**
3. Name it something like: class-mini-project
4. Add a description (e.g., "Simple HTML project")
5. Check **“Initialize with a README”**
6. Click **Create repository**

**Step 3: Invite Your Classmate to Collaborate**

1. Go to your repo → Click **Settings**
2. In the left menu, click **"Collaborators"**
3. Click **"Add people"**
4. Type your classmate’s GitHub **username or email**
5. Click **Add** → Your classmate will get an **email invitation**
6. They must **accept the invite** to start collaborating

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

**1. System Software**

These help the computer run and manage itself.

| **Software Name** |  | **Type** |
| --- | --- | --- |
| Windows 10 / 11 |  | System Software |
| macOS |  | System Software |

**2. Application Software**

These are the programs you use to do tasks (writing, browsing, etc.).

| **Software Name** | **Type** |
| --- | --- |
| Microsoft Word | Application Software |
| Google Chrome | Application Software |
| WhatsApp | Application Software |
| VLC Media Player | Application Software |

**3. Utility Software**

These help maintain, protect, and improve your system.

| **Software Name** | **Type** |
| --- | --- |
| Antivirus (e.g., Avast) | Utility Software |
| WinRAR / 7-Zip | Utility Software |
| Disk Cleanup (Windows) | Utility Software |
| CCleaner | Utility Software |

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

| **Task** | **Command** |
| --- | --- |
| Clone | git clone [repo link] |
| New Branch | git branch new-feature |
| Switch Branch | git checkout new-feature |
| Add Changes | git add . |
| Commit Changes | git commit -m "message" |
| Merge | git merge new-feature |
| Push | git push origin main |

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

**1. Word Processing Software**

**Example:** Microsoft Word, Google Docs  
**Use:** Helps in writing and editing documents.  
**Productivity Boost:**

* Fast typing and formatting of documents
* Spell check and grammar suggestions
* Easy to share and collaborate

**2. Spreadsheet Software**

**Example:** Microsoft Excel, Google Sheets  
**Use:** Data calculation, analysis, and graphing  
**Productivity Boost:**

* Automates complex calculations
* Visualizes data through charts
* Helps in budgeting and financial planning

**3. Presentation Software**

**Example:** Microsoft PowerPoint, Google Slides  
**Use:** Create slide-based presentations  
**Productivity Boost:**

* Easy way to share ideas visually
* Supports multimedia (images, videos, etc.)
* Used in meetings, classes, and trainings

**4. Database Management Software**

**Example:** Microsoft Access, MySQL  
**Use:** Store, manage, and retrieve data  
**Productivity Boost:**

* Organizes large volumes of data efficiently
* Allows quick data searching and filtering
* Supports multi-user access

**5. Web Browsers**

**Example:** Google Chrome, Mozilla Firefox  
**Use:** Accessing and browsing the internet  
**Productivity Boost:**

* Quick access to online tools and resources
* Extensions and bookmarks save time
* Helps with research and communication

**6. Email Clients**

**Example:** Microsoft Outlook, Gmail  
**Use:** Sending and receiving emails  
**Productivity Boost:**

* Fast communication
* Calendar and task features
* Organizes messages for better workflow

**7. Graphic Design Software**

**Example:** Adobe Photoshop, Canva  
**Use:** Creating and editing images or designs  
**Productivity Boost:**

* Helps in professional content creation
* Speeds up the design process
* Supports brand and marketing tasks

**8. Communication Software**

**Example:** Zoom, Microsoft Teams, Slack  
**Use:** Video calls, chat, and collaboration  
**Productivity Boost:**

* Enables remote work and online meetings
* Improves team collaboration
* Supports file sharing and screen sharing

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

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

**Requirement Specification: Library Management System**

**1. Introduction**

The Library Management System (LMS) is a software designed to manage the daily activities of a library like book issuing, returns, member registration, and inventory management.

**2. Objectives**

* To store book details like title, author, ISBN, etc.
* To manage student and staff members.
* To track books that are issued and returned.
* To calculate fines for late returns.
* To generate reports of book usage and availability.

**3. Functional Requirements**

**Book Management**

* Add, update, delete book records.
* Search books by title, author, or ISBN.
* View available or issued books.

**Member Management**

* Register new members (students/staff).
* Update member information.
* View borrowing history of members.

**Issue and Return System**

* Issue a book to a member.
* Record book return.
* Check for due dates and calculate fines.

**Report Generation**

* List of issued books.
* List of overdue books.
* Fine collection report.
* Inventory report.

**4. Non-Functional Requirements**

* **Database:** All data will be stored securely in a relational database.
* **Security:** Only authorized users (librarian/admin) can modify records.
* **User Interface:** Easy-to-use interface with forms and menus.
* **Performance:** The system should respond quickly (within 2 seconds for queries).

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

**1. User Management**

* **Register Account**: New users can sign up with name, email, password, etc.
* **Login/Logout**: Users can log in securely and log out anytime.
* **Update Profile**: Users can update personal details and shipping address.
* **Forgot Password**: Users can reset forgotten passwords via email.

**2. Product Catalog**

* **View Products**: Users can browse products by categories (e.g., electronics, clothes, etc.).
* **Search Products**: Users can search using keywords or filters (price, rating, brand).
* **View Product Details**: Clicking a product shows more info (description, images, price, reviews).

**3. Shopping Cart**

* **Add to Cart**: Users can add items to their cart.
* **Remove from Cart**: Users can remove unwanted items.
* **Update Quantity**: Users can increase or decrease product quantity in cart.
* **View Cart**: Shows list of selected items with total price.

**4. Checkout and Payment**

* **Place Order**: Users proceed to buy products in the cart.
* **Enter Shipping Info**: Users input shipping address and contact info.
* **Payment Options**: Credit card, debit card, UPI, wallet, etc.
* **Order Confirmation**: User receives confirmation message and order number.

**5. Order Management**

* **View Order History**: Users can see past orders with details.
* **Track Orders**: Users can track shipping status.
* **Cancel Order**: Option to cancel an order before it’s shipped.
* **Return/Exchange**: Request return or replacement if allowed.

**6. Review & Rating System**

* **Leave Review**: Users can write reviews after receiving a product.
* **Rate Product**: Users can give a star rating (1 to 5).
* **View Other Reviews**: Users can read feedback from other buyers.

**7. Admin Functions**

* **Manage Products**: Add, update, or remove products.
* **Manage Orders**: View, approve, or cancel user orders.
* **Manage Users**: View user info or deactivate accounts.
* **Generate Reports**: Sales, top products, order summary, etc.

**Summary**

The Online Shopping System provides:

* A user-friendly shopping experience
* Easy product browsing and filtering
* Secure checkout and payment
* Order tracking and management

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

**Basic System Architecture for a Food Delivery App**

We'll divide the system into **three main layers**:

**1. Frontend (User Interface)**

The part users interact with — different interfaces for:

* **Customer App**
  + Browse restaurants and menus
  + Place orders
  + Track delivery
  + Make payments
* **Restaurant App**
  + Receive and manage orders
  + Update menu and prices
* **Delivery Agent App**
  + View assigned deliveries
  + Navigate to pickup and drop-off locations

**2. Backend (Application Server/Business Logic)**

This is where the main operations happen — manages communication between apps and the database

* **Authentication Service**
  + Log in/Sign up (customers, restaurants, agents)
* **Order Management System**
  + Handles placing, updating, and tracking orders
* **Restaurant Service**
  + Maintains restaurant info, menu, availability
* **Location Service**
  + Tracks delivery agent's live location
* **Payment Gateway Integration**
  + Processes payments securely
* **Notification System**
  + Sends updates (SMS, push notifications)

**3. Database Layer (Data Storage)**

Stores all the data required for smooth operation:

* Users: Customer, Restaurant, and Delivery Agent profiles
* Menus: Dishes, prices, restaurant availability
* Orders: Order status, history, ratings
* Payments: Transactions and billing
* Locations: Addresses, coordinates

**Flow of Operation (Example)**

1. **Customer** opens the app and selects food items.
2. **Order** is sent to the backend and stored in the database.
3. **Restaurant** gets the order and accepts it.
4. Backend assigns a **Delivery Agent**.
5. Agent **picks up** the food and delivers it using **navigation service**.
6. Customer **pays** online or on delivery.
7. System sends **real-time notifications** at each step.

**Develop test cases for a simple calculator program**

Here’s a **simple and clear list of test cases** for a basic calculator program that can perform:

Addition  
 Subtraction  
Multiplication  
 Division

We'll cover both **positive** and **edge** (tricky) cases.

**Assumptions**

* Calculator takes **two inputs** (numbers)
* User selects an **operation**: +, -, ×, ÷
* Outputs a **single result**

**Test Cases for Calculator**

| **Test Case ID** | **Description** | **Input 1** | **Input 2** | **Operation** | **Expected Result** |
| --- | --- | --- | --- | --- | --- |
| TC01 | Add two positive numbers | 5 | 3 | + | 8 |
| TC02 | Add two negative numbers | -4 | -6 | + | -10 |
| TC03 | Add positive and negative | 7 | -2 | + | 5 |
| TC04 | Subtract smaller from bigger | 10 | 4 | - | 6 |
| TC05 | Subtract bigger from smaller | 3 | 7 | - | -4 |
| TC06 | Multiply two numbers | 6 | 3 | \* | 18 |
| TC07 | Multiply by 0 | 9 | 0 | \* | 0 |
| TC08 | Divide two numbers | 8 | 2 | / | 4 |
| TC09 | Divide by 0 (invalid) | 5 | 0 | / | Error / Exception |

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

**Real-World Case Study: WhatsApp Global Outage (October 2022)**

**Overview**

On **October 25, 2022**, **WhatsApp**, one of the most used messaging apps in the world (owned by Meta), experienced a major **global outage**. The service was unavailable for over 2 hours across multiple regions including India, Europe, and parts of the US.

**Problem Description**

* Users were unable to:
  + Send or receive messages
  + Create or update group chats
  + Access features like media sharing and calls
* Outage was not device-specific — both **Android** and **iOS** users were affected.
* **WhatsApp Web** was also down.

**Cause of the Problem**

Meta later stated that the issue was due to a **configuration change** on their backend servers. This type of change can:

* Break communication between systems
* Crash databases or services
* Cause requests to time out

**Maintenance Actions Taken**

* WhatsApp’s engineering team immediately began investigating the backend logs.
* They identified the misconfiguration in one of the **core routing services**.
* Engineers **rolled back** the recent changes.
* After rollback, they monitored traffic to ensure stability.
* The full service was gradually restored region by region.

**Outcome**

* The issue was resolved in about **2 hours**.
* WhatsApp issued a public apology and explanation.
* The incident raised awareness about:
  + The importance of **change testing**
  + **Failover systems**
  + Need for **monitoring tools** to detect early signs of failure

**Lesson Learned**

* Always test configuration changes in **staging environments** before pushing to production.
* Use **version control and rollback plans**.
* Implement **health checks** and **alerts** for real-time monitoring.
* Have a **disaster recovery plan** in place.

**Create a DFD for a hospital management system.**

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