**MERN STACK LAB EXERCISE**

# **Module 1 – SE - Overview of IT Industry**

**Q>** **Write a simple "Hello World" program in two different programming languages**

**of your choice. Compare the structure and syntax.**

Ans> Certainly, here's a simple "Hello World" program in two popular languages:

1. Python:

Python

print("Hello, World!")

2. JavaScript:

JavaScript

console.log("Hello, World!");

Comparison:

* Structure:
  + Python:
    - Relies on the print() function to display output to the console.
    - Uses indentation to define code blocks.
  + JavaScript:
    - Utilizes the console.log() method to print output to the browser console.
    - Employs curly braces {} to enclose code blocks and semicolons ; to terminate statements.
* Syntax:
  + Python: Generally more concise and readable, often using English-like keywords.
  + JavaScript: More verbose and requires stricter adherence to syntax rules, including semicolons.

Key Differences:

* Typing: Python is dynamically typed, while JavaScript is dynamically and weakly typed.
* Purpose: Python is a versatile language used for various purposes (data science, web development, scripting), while JavaScript is primarily used for front-end web development.

This simple example demonstrates the basic structure and syntax differences between Python and JavaScript. These differences can significantly impact how you approach programming tasks in each language.

**Q>** **: Research and create a diagram of how data is transmitted from a client to a server over the internet.**

Ans>

[ Client ]

|

| 1. Sends Request (HTTP, DNS Lookup)

|

v

[ Internet / Network ]

|

| 2. Data routed through Internet Backbone

|

v

[ Server ]

|

| 3. Server processes request, sends back response

|

v

[ Internet / Network ]

|

| 4. Data routed back to Client

|

v

[ Client ]

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

Ans> Here's a simple HTTP client-server communication example using Python. This demonstrates a basic GET request and response.

Server.py

from http.server import BaseHTTPRequestHandler, HTTPServer

class MyHandler(BaseHTTPRequestHandler):

def do\_GET(self):

self.send\_response(200) # Send OK response

self.send\_header('Content-type', 'text/html')

self.end\_headers()

message = "<html><body><h1>Hello from the server!</h1><p>You requested path: " + self.path + "</p></body></html>"

self.wfile.write(message.encode()) # Send message

def run(server\_class=HTTPServer, handler\_class=MyHandler, port=8000):

server\_address = ('', port) # '' means listen on all interfaces

httpd = server\_class(server\_address, handler\_class)

print(f"Starting httpd on port {port}...")

try:

httpd.serve\_forever()

except KeyboardInterrupt:

pass

httpd.server\_close()

print("Stopping httpd...")

if \_\_name\_\_ == '\_\_main\_\_':

run()

Client.py

import requests

def make\_request(url):

try:

response = requests.get(url)

response.raise\_for\_status() # Raise an exception for bad status codes (4xx or 5xx)

print("Status Code:", response.status\_code)

print("Response Content:")

print(response.text) # Print the HTML content

return response.text

except requests.exceptions.RequestException as e:

print(f"Error: {e}")

return None

if \_\_name\_\_ == "\_\_main\_\_":

server\_url = "http://localhost:8000/some/path" # Example URL

make\_request(server\_url)

server\_url\_root = "http://localhost:8000" # Example root URL

make\_request(server\_url\_root)

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

Ans> You got it! Here's a breakdown of internet connection types with fewer words:

**DSL (Phone Line):**

* **Pros:** Widely available, cheap.
* **Cons:** Slow, speed drops with distance.

**Cable (Coaxial Cable):**

* **Pros:** Faster than DSL, widely available.
* **Cons:** Congestion slows speeds, slower uploads.

**Fiber (Fiber Optic):**

* **Pros:** Fastest, reliable, low latency.
* **Cons:** Limited availability, costly.

**Satellite (Satellite Dish):**

* **Pros:** Available anywhere.
* **Cons:** Slow, high latency, weather issues.

**Fixed Wireless (Radio Waves):**

* **Pros:** Good for rural areas.
* **Cons:** Affected by distance/obstacles/weather.

**Mobile Broadband (Cellular):**

* **Pros:** Mobile, convenient.
* **Cons:** Data caps, speed varies, can be pricey**.**

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

Ans> Let's simulate HTTP and FTP requests using command-line tools. curl is excellent for HTTP, while ftp (or curl with appropriate options) can handle FTP.

**HTTP (using curl):**

* **GET Request:** Retrieves data from a URL.

Bash

curl https://www.example.com

This will fetch the HTML content of www.example.com and display it in your terminal.

To save the output to a file:

Bash

curl -o index.html https://www.example.com

Or:

Bash

curl https://www.example.com > index.html

* **GET Request with Headers:**

Bash

curl -H "User-Agent: MyCustomUserAgent" https://www.example.com

This adds a custom User-Agent header to the request.

* **POST Request:** Sends data to a server.

Bash

curl -X POST -H "Content-Type: application/json" -d '{"name": "John Doe", "age": 30}' https://httpbin.org/post

* + -X POST: Specifies the POST method.
  + -H "Content-Type: application/json": Sets the content type of the data being sent.
  + -d '{"name": "John Doe", "age": 30}': The data to be sent (in JSON format in this example).
  + https://httpbin.org/post: A handy website for testing HTTP requests. It will echo back the data you send.
* **Getting only the HTTP Headers:**

Bash

curl -I https://www.example.com

This will return only the HTTP headers from the server's response, without the page content.

* **Following Redirects:** Some websites redirect to other URLs. To follow these redirects:

Bash

curl -L https://httpstat.us/301 # Example of a 301 redirect

The -L option tells curl to follow redirects.

**FTP (using ftp or curl):**

* **Connecting to an FTP Server (using ftp):**

Bash

ftp ftp.example.com

This will open an interactive FTP session. You'll then be prompted for a username and password (or you can often use anonymous as the username and your email as the password for public FTP servers). Once connected, you can use FTP commands like ls (list files), get (download a file), put (upload a file), cd (change directory), etc.

* **Downloading a File (using curl):**

Bash

curl ftp://user:password@ftp.example.com/path/to/file.txt -o file.txt

* + ftp://user:password@ftp.example.com/path/to/file.txt: The FTP URL, including username and password. Replace with your actual credentials and path.
  + -o file.txt: Saves the downloaded file as file.txt.
* **Uploading a File (using curl):**

Bash

curl -T local\_file.txt ftp://user:password@ftp.example.com/path/to/upload/

* + -T local\_file.txt: Specifies the local file to upload.

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

Ans> Okay, here's a more concise yet detailed explanation of three common application security vulnerabilities:

**1. SQL Injection (SQLi):**

* **What:** Malicious SQL code injected into user inputs, allowing attackers to manipulate database queries.
* **How:** Improperly sanitized input is concatenated into SQL queries, enabling attackers to bypass authentication, steal data, or execute commands.
* **Solution:** *Parameterized queries (prepared statements)* are the most effective defense. They treat user input as data, not code, preventing injection. Input validation and least privilege are also important.

**2. Cross-Site Scripting (XSS):**

* **What:** Injecting malicious scripts (usually JavaScript) into websites viewed by other users.
* **How:** Unsanitized user input is displayed on web pages, allowing injected scripts to execute in victims' browsers, stealing cookies, session tokens, or redirecting to malicious sites.
* **Solution:** *Output encoding/escaping* converts special characters to HTML entities, preventing them from being interpreted as code. *Content Security Policy (CSP)* controls which resources a browser can load. Input validation and HttpOnly cookies are also helpful.

**3. Cross-Site Request Forgery (CSRF):**

* **What:** Tricking a logged-in user into performing unwanted actions on a website they're authenticated to.
* **How:** Attackers craft malicious requests (links, images) that, when accessed by the victim, are automatically sent to the target website with the user's cookies, effectively performing actions on their behalf.
* **Solution:** *Anti-CSRF tokens* (unique, unpredictable tokens per session) are the primary defense. *SameSite cookies* prevent cookies from being sent with cross-site requests. Double submit cookies are another option.

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

Ans> Here are 5 common applications classified as system or application software:

* **Operating System (Windows, macOS):** *System*
* **Web Browser (Chrome, Safari):** *Application*
* **Text Editor (Notepad, VS Code):** *Application*
* **Database (MySQL, SQL Server):** *System*
* **Email Client (Outlook, Gmail):** *Application*

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

Ans>

**Summary Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Functionality | Example Actions in Online Shopping System | Technologies |
| Presentation | User interface; displays information and captures user input. | Search bar, product display, "Add to Cart" button, shopping cart page, checkout forms. | HTML, CSS, JavaScript, Front-end frameworks (React, Angular) |
| Business Logic | Core application logic; processes user requests, enforces business rules, interacts with the data access layer. | Processing search queries, managing shopping carts, handling checkout process (validations, payment processing, inventory updates, order confirmation). | Server-side languages (Python, Java, Node.js), Application frameworks (Django, Spring, Express.js) |
| Data Access | Interacts with the database; retrieves and stores data; isolates business logic from database implementation. | Executing SQL queries to retrieve book data, update inventory, retrieve user information. | Database (MySQL, PostgreSQL), ORM (SQLAlchemy, Hibernate) or direct SQL. |

This case study demonstrates how the three layers work together in a typical web application. Each layer has a specific responsibility, contributing to the overall functionality and maintainability of the system.

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

Ans> **Software Environments:**

* **Development:** Local coding, testing, debugging.
* **Testing (QA/Staging):** Mirrors production, tests functionality.
* **Production (Live):** End-user environment, focus on stability.

**Setting up a VM (VirtualBox & Ubuntu):**

1. Install VirtualBox.
2. Download Ubuntu Server ISO.
3. Create new VM (select Linux/Ubuntu).
4. Configure VM (storage: ISO, network: Bridged).
5. Start VM, install Ubuntu (enable SSH).
6. Update: sudo apt update && sudo apt upgrade -y
7. Install tools (e.g., sudo apt install python3 -y).
8. Access via console or SSH (ssh user@ip).

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

Ans> I can't directly interact with GitHub or any external systems like a human user can. I can't create accounts, push commits, or perform other actions that require real-time interaction with online services.

However, I *can* provide you with the steps and an example of a simple Python program that you can then use to create your first GitHub repository and upload your code.

**Steps to Create a GitHub Repository and Upload Code:**

1. **Create a GitHub Account (if you don't have one):** Go to [github.com](https://www.google.com/url?sa=E&source=gmail&q=github.com) and sign up.
2. **Create a New Repository:**
   * Log in to GitHub.
   * Click the "+" button in the top right corner and select "New repository."
   * Give your repository a name (e.g., "my-first-repo").
   * You can add a description (optional).
   * Choose whether you want the repository to be public or private.
   * You can initialize the repository with a README file (recommended). This will create a basic file explaining your project.
   * Click "Create repository."
3. **Create Your Source Code File (locally):**
   * Create a new file on your computer using a text editor or IDE. Let's create a simple Python file named hello.py:

<!-- end list -->

Python

def greet(name):

"""Greets the person passed in as a parameter."""

print(f"Hello, {name}!")

if \_\_name\_\_ == "\_\_main\_\_":

user\_name = input("What's your name? ")

greet(user\_name)

1. **Open a Terminal or Git Bash:** Navigate to the directory where you saved hello.py.
2. **Initialize a Git Repository (locally):**

Bash

git init

1. **Add Your File to the Staging Area:**

Bash

git add hello.py

If you have other files you want to add as well, you can use git add . to add all files in the current directory.

1. **Commit Your Changes:**

Bash

git commit -m "Initial commit: Added hello.py"

The -m flag specifies the commit message.

1. **Connect Your Local Repository to the Remote GitHub Repository:**

You'll need the URL of your GitHub repository. You can find it on the repository's page on GitHub. It will look something like this: https://github.com/YourUsername/your-repo-name.git or git@github.com:YourUsername/your-repo-name.git (SSH URL). Use one of the following commands based on if you want to use HTTPS or SSH:

**HTTPS:**

Bash

git remote add origin https://github.com/YourUsername/your-repo-name.git

**SSH:**

Bash

git remote add origin git@github.com:YourUsername/your-repo-name.git

1. **Push Your Code to GitHub:**

Bash

git push -u origin main

* + -u sets the upstream branch, so you only need to use git push in the future.
  + origin is the name of the remote repository (which we set in the previous step).
  + main is the name of the branch you're pushing to (the default branch name on GitHub).

You'll likely be prompted for your GitHub username and password (or SSH key passphrase if you set up SSH).

Now, if you go to your GitHub repository in your web browser, you should see your hello.py file.

This is the basic workflow for creating a repository and uploading code to GitHub. There are more advanced Git features (branching, merging, etc.), but this gets you started.

**Q> Create a Github repository and document how to commit and push code changes.**

Ans> I can't directly create a GitHub repository. I'm a language model, not a user with access to the internet or GitHub. However, I *can* give you the steps, and then you can follow them on GitHub.

Steps to Create a Repo and Commit/Push Changes:

1. Create a GitHub Account (if you don't have one): Go to github.com and sign up.
2. Create a New Repository:
   * Log in to GitHub.
   * Click "+" (top right) -> "New repository".
   * Name your repo (e.g., "my-repo"). Add a description (optional).
   * Choose Public or Private. Initialize with a README (recommended).
   * Click "Create repository".
3. Local Setup (in your terminal/Git Bash):
   * git clone <repository\_url> (Copy the URL from your GitHub repo page). This downloads the empty repo to your computer.
   * Navigate into the downloaded directory: cd my-repo (or whatever you named the repo).
4. Make Changes (create/edit files):
   * Create or edit files in the repository directory (e.g., create my\_file.txt).
5. Commit Changes:
   * git add . (Adds all changes) or git add my\_file.txt (adds a specific file).
   * git commit -m "Your commit message" (e.g., git commit -m "Added my first file").
6. Push Changes to GitHub:
   * git push origin main (This uploads your commits to the main branch on GitHub).

Example:

Let's say you created a file hello.txt with the content "Hello, world!".

Bash

git add hello.txt

git commit -m "Added hello.txt file"

git push origin main

That's it! You've created a repo, committed changes, and pushed them to GitHub.

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

Ans> Here's a list of commonly used software categorized by type:

**System Software:**

* Operating Systems (Windows, macOS, Linux)
* Device Drivers (Printer drivers, Graphics drivers)
* Firmware (BIOS, UEFI)

**Application Software:**

* Web Browsers (Chrome, Firefox, Safari)
* Office Suites (Microsoft Office, Google Workspace)
* Image Editors (Photoshop, GIMP)
* Video Players (VLC, Media Player)
* Games

**Utility Software:**

* Antivirus Software (Windows Defender, McAfee)
* File Compression Tools (7-Zip, WinRAR)
* Disk Cleanup Tools (Disk Cleanup on Windows)
* System Monitoring Tools (Task Manager, Activity Monitor)

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

Ans> Here's a list of commonly used software categorized by type:

**System Software:**

* Operating Systems (Windows, macOS, Linux)
* Device Drivers (Printer drivers, Graphics drivers)
* Firmware (BIOS, UEFI)

**Application Software:**

* Web Browsers (Chrome, Firefox, Safari)
* Office Suites (Microsoft Office, Google Workspace)
* Image Editors (Photoshop, GIMP)
* Video Players (VLC, Media Player)
* Games

**Utility Software:**

* Antivirus Software (Windows Defender, McAfee)
* File Compression Tools (7-Zip, WinRAR)
* Disk Cleanup Tools (Disk Cleanup on Windows)
* System Monitoring Tools (Task Manager, Activity Monitor)

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

Ans> **Report: Types of Application Software and Productivity**

Application software is designed for specific user tasks. Its diverse categories directly enhance productivity in various ways. Here's a summary:

**1. Productivity Software:**

* **Examples:** Word processors (Microsoft Word), spreadsheets (Excel), presentation software (PowerPoint), project management tools (Trello, Asana).
* **Productivity Gains:** Streamline document creation, data analysis, presentations, and task management, improving efficiency and organization.

**2. Communication Software:**

* **Examples:** Email clients (Outlook, Gmail), instant messaging (Slack, Microsoft Teams), video conferencing (Zoom, Google Meet).
* **Productivity Gains:** Facilitate communication, collaboration, and information sharing, reducing delays and improving teamwork.

**3. Graphics and Multimedia Software:**

* **Examples:** Image editors (Photoshop, GIMP), video editors (Adobe Premiere Pro), audio editors (Audacity).
* **Productivity Gains:** Enable creation and manipulation of visual and audio content for marketing, design, and other creative tasks.

**4. Database Software:**

* **Examples:** Database management systems (MySQL, PostgreSQL), database applications (Access).
* **Productivity Gains:** Organize, store, and retrieve large amounts of data efficiently, supporting data-driven decision-making.

**5. Business Software:**

* **Examples:** Customer relationship management (CRM) systems (Salesforce), enterprise resource planning (ERP) systems (SAP).
* **Productivity Gains:** Automate business processes, manage customer interactions, and improve overall business operations.

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

Ans> Code snippet

graph TD

A[Planning & Requirements Gathering] --> B{Analysis & Design};

B --> C[Implementation (Coding)];

C --> D{Testing};

D -- Tests Passed --> E[Deployment];

D -- Tests Failed --> C;

E --> F[Maintenance & Evolution];

F --> G{End of Life/Retirement};

This flowchart represents a simplified version of the SDLC. There are various SDLC models (Waterfall, Agile, Spiral, etc.), each with its own specific steps and processes. However, this flowchart captures the essential phases present in most models.

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

Ans> **Library Management System Requirements**

**Goals:** Manage books, members, and borrowing/returning efficiently.

**Users:** Librarians, members.

**Functional Requirements:**

* **Book Management:** Add, search (by title, author, ISBN), view details, update, remove (librarian only).
* **Member Management:** Add, search (by ID, name), view details, update, remove (librarian only).
* **Borrowing/Returning:** Record borrowing/return (with dates), check availability, generate overdue reports.
* **Reporting:** List all books/members, borrowed books report.

**Non-Functional Requirements:**

* Performance: Responsive, quick search.
* Security: Restricted access to sensitive data.
* Usability: Easy to use.
* Availability: During library hours.
* Data Storage: Persistent data.

**Key Data:**

* Books: Title, author, ISBN, publisher, year, copies, genre.
* Members: ID, name, address, contact.

**Open Issues:** Loan period, overdue fines.

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

Ans> **Online Shopping System Functional Analysis:**

**User-Focused:**

* **Browsing/Searching:** Find products (search, filter, sort).
* **Shopping Cart:** Manage selected items (add, view, update, remove).
* **Checkout:** Complete purchase (shipping, billing, payment, confirmation).
* **User Accounts:** Manage profile and orders.
* **Customer Support:** Get assistance (contact, FAQ).

**System-Focused:**

* **Inventory:** Track product stock.
* **Order Management:** Process and fulfill orders.
* **Payment Integration:** Secure transactions.
* **Reporting:** Analyze sales and data.

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

Ans>

+---------------------+ +----------------------+

| | | |

| User | | Restaurant |

| | | Management |

+---------+----------+ +----------+----------+

| |

v v

+-------------------+ +---------------------------+

| User Profile | | Menu Management|

| & Ordering | <------>| & Order Mgmt |

+-------------------+ +---------------------------+

| |

v v

+---------------------+ +----------------------+

|Payment System| | Order Processing|

+---------------------+ +----------------------+

^ |

| v

+---------+----------+ +---------------------+

| | | Delivery Person|

| Admin | | Assignment |

| | +---------------------+

+--------------------+

This is a basic architecture. Real-world food delivery apps are often more complex, with additional components for analytics, promotions, reviews, etc.

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

Ans> **Case: Healthcare System Outage (Example)**

**Problem:** A hospital's patient record system experienced a critical database failure. This prevented doctors and nurses from accessing patient medical histories, lab results, and medication information.

**Impact:**

* Delayed diagnoses and treatments.
* Increased risk of medical errors.
* Disrupted hospital operations.

**Maintenance Required:**

* Immediate data recovery from backups.
* Database repair and optimization.
* Root cause analysis to prevent recurrence (e.g., hardware failure, software bug).
* Implementation of improved backup and disaster recovery procedures.

**Outcome:** System was restored, but the outage highlighted the critical need for robust maintenance and disaster recovery planning in healthcare software.

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

Ans> +---------------------+ +----------------------+

| | | |

| User | | Restaurant |

| | | Management |

+---------+----------+ +----------+----------+

| |

v v

+-------------------+ +---------------------------+

| User Profile | | Menu Management|

| & Ordering | <------>| & Order Mgmt |

+-------------------+ +---------------------------+

| |

v v

+---------------------+ +----------------------+

|Payment System| | Order Processing|

+---------------------+ +----------------------+

^ |

| v

+---------+----------+ +---------------------+

| | | Delivery Person|

| Admin | | Assignment |

| | +---------------------+

+--------------------+

This DFD provides a high-level overview of the data flow within a hospital management system. It can be further decomposed into more detailed diagrams for each process if necessary. This simplified version focuses on the core data flows and components for better understanding.

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

Ans> +---------------------+ +-------------------+

| | | |

| User | | Admin |

| Input Data | | Verification |

+---------+-----------+ +--------+----------+

| |

| v

v +-----------------------+

+-------------------+ | Verify Input |

| Display Form |<----- | (Validation) |

+-------------------+ +-----------+-----------+

| |

v v

+---------------------------+ +-------------------------+

| Enter Details (Name,| | Valid? (Yes/No) |

| Email, Password) | +------------+------------+

+---------------------------+ |

| v

v +-----------------+

+--------------------------+ | Registration |

|Submit Registration | | Successful |

+--------------------------+ +------------------+

| |

v v

+------------------------------+ +---------------------------+

| Confirmation Message| | Send Confirmation |

| (Email/SMS) | | Email/SMS to User |

+------------------------------+ +---------------------------+

| |

v v

+-------------------+ +------------------+

| End Process | | End Process |

+-------------------+ +------------------+