**1.Write A Simple "Hello World" Program In Two Different Programming Languages Of Your Choice. Compare The Structure And Syntax.**

Here’s A Simple **"Hello World"** Program In Two Popular Programming Languages: **Python** And **C**.

Python Code:

Print("Hello, World!");

C Code:

#Include <Stdio.H>

Int Main() {

Printf("Hello, World!\N");

Return 0;

}

Comparison Of Structure And Syntax:

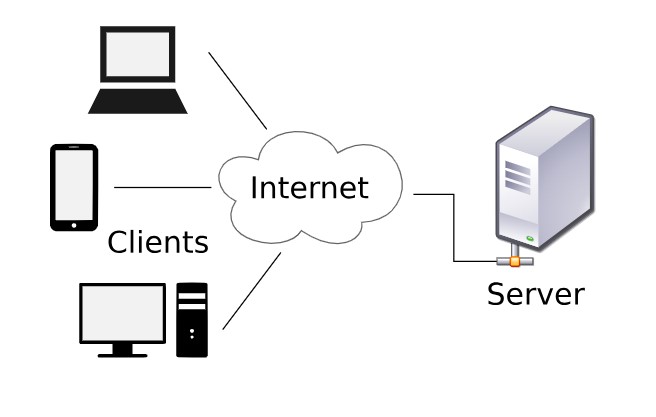
**Syntax**:

* **Python**: Python Uses A Simple And Clean Syntax. It Doesn’t Require The Use Of Semicolons To Terminate Statements And Doesn't Need Explicit Declaration Of The Main Function. The Code Is Written In A More Human-Readable Format.
* **C**: C Has A More Complex Syntax. It Requires #Include For Including Libraries, The Use Of A Main Function (Int Main()), And Semicolons To Terminate Statements.

**Program Structure**:

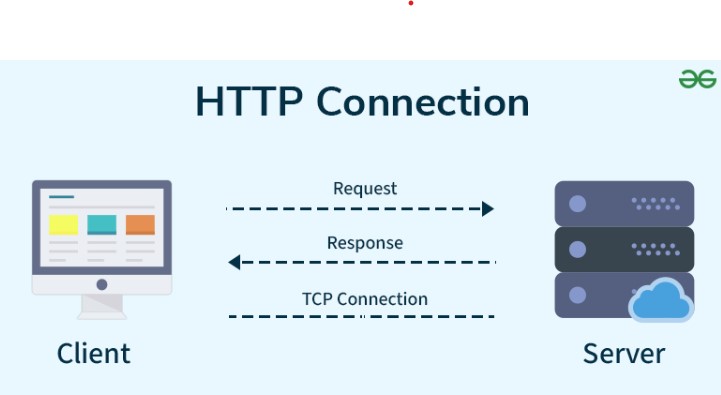
* **Python**: The Program Consists Of A Single Line Of Code, And Python Does Not Require A Main Function To Execute.
* **C**: C Programs Require A Main() Function To Serve As The Entry Point, And Return Values (Such As Return 0;) Are Commonly Used.

**2. Research And Create A Diagram Of How Data Is Transmitted From A Client To A Server Over The Internet.**



Data Transmission From A Client To A Server Involves Sending A Request Over The Internet Through Various Routers, With DNS Resolving The Server’s IP. At The Transport Layer, Data Is Split Into Packets For Reliable Transmission. The Server Processes The Request And Sends A Response Back Through The Same Steps. In Short, The Process Ensures Efficient Communication Across Different Layers Of The OSI Model.

**3. Design A Simple HTTP Client-Server Communication In Any Language.**



In HTTP Client-Server Communication, The Client Sends A Request To The Server, Typically Over The Internet, And The Server Responds With Data.

1. **Client Initiates The Request**: The Client, Usually A Web Browser Or Application, Sends An HTTP Request To The Server. This Request Specifies The Resource It Wants (E.G., A Web Page, Image, Or Data).
2. **DNS Resolution**: The Client First Resolves The Server’s Domain Name (Like Www.Example.Com) Into An IP Address Using The Domain Name System (DNS), Allowing It To Locate The Server.
3. **Server Receives The Request**: The Server Listens For Incoming Requests On A Specific Port (Usually Port 80 For HTTP). Upon Receiving The Request, It Processes It Based On The Type Of HTTP Method (GET, POST, Etc.).
4. **Server Sends A Response**: The Server Responds To The Client’s Request With The Appropriate Data, Such As A Web Page, JSON, Or An Error Message If The Request Can't Be Processed.
5. **Client Receives The Response**: The Client Processes The Server’s Response And Displays It, Often Rendering Web Pages Or Using The Data In Applications.

**4. Research Different Types Of Internet Connections (E.G., Broadband, Fiber, Satellite) And List Their Pros And Cons.**

Here’s A Quick Overview Of Different Types Of Internet Connections And Their Pros And Cons:

**1. Fiber-Optic Internet:**

* **Pros:** Very High Speeds, Low Latency, Reliable.
* **Cons:** Limited Availability, Expensive Infrastructure.

**2. Cable Internet:**

* **Pros:** High Speeds, Widely Available In Urban Areas.
* **Cons:** Shared Bandwidth Can Slow Speeds During Peak Usage.

**3. DSL (Digital Subscriber Line):**

* **Pros:** Widely Available, Stable Connection.
* **Cons:** Slower Speeds Compared To Fiber, Performance Drops With Distance From The Provider.

**4. Satellite Internet:**

* **Pros:** Available In Remote Areas, Easy Setup.
* **Cons:** High Latency, Variable Speeds, Affected By Weather.

**5. Fixed Wireless Internet:**

* **Pros:** Quick Setup, Good For Rural Areas.
* **Cons:** Requires Line-Of-Sight, Speeds Can Vary With Environmental Factors.

**6. Mobile Broadband (3G/4G/5G):**

* **Pros:** Portable, Convenient For On-The-Go Access.
* **Cons:** Data Limits, Speeds Vary Based On Location.

**7. Dial-Up Internet:**

* **Pros:** Available Anywhere With A Phone Line, Low Cost.
* **Cons:** Very Slow Speeds, Uses Phone Line For Internet.

**8. Broadband Over Power Lines (BPL):**

* **Pros:** Uses Existing Power Lines, Potential For Decent Speeds.
* **Cons:** Limited Availability, Electrical Interference Can Disrupt Signal.

**5. Simulate HTTP And FTP Requests Using Command Line Tools (E.G., Curl).**

Here’s How You Can Simulate HTTP And FTP Requests Using Curl From The Command Line:

**1. HTTP Request (GET Method)**

Curl Http://Example.Com

* This Sends A GET Request To Http://Example.Com And Retrieves The Content Of The Webpage.

**2. HTTP Request With POST Method**

Curl -X POST Http://Example.Com -D "Name=John&Age=30"

* This Sends A POST Request With Data (Name=John&Age=30) To The Specified URL.

**3. FTP Request (Download A File)**

Curl -U Username:Password Ftp://Example.Com/Path/To/File -O Localfile.Txt

* This Downloads A File From An FTP Server Using The Specified Credentials And Saves It As Localfile.Txt.

**4. FTP Request (Upload A File)**

Curl -U Username:Password -T Localfile.Txt Ftp://Example.Com/Path/To/Remote/File

* This Uploads Localfile.Txt To The FTP Server.

**6. Identify And Explain Three Common Application Security Vulnerabilities. Suggest Possible Solutions.**

Here Are Three Common Application Security Vulnerabilities And Their Solutions:

**1. SQL Injection**

* **Explanation:** Attackers Insert Malicious SQL Queries Into Input Fields To Manipulate Databases.
* **Solution:** Use Prepared Statements And Parameterized Queries To Safely Handle User Input.

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

* **Explanation:** Malicious Scripts Are Injected Into Web Pages Viewed By Other Users, Potentially Stealing Sensitive Information.
* **Solution:** Sanitize And Validate All User Inputs, And Use Content Security Policy (CSP) Headers To Restrict Script Execution.

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

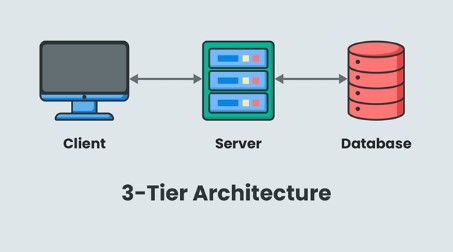
* **Explanation:** Attackers Trick Users Into Performing Actions On A Website Without Their Consent.
* **Solution:** Use Anti-CSRF Tokens To Ensure Requests Are Coming From Authenticated Users, And Require Re-Authentication For Sensitive Actions.

**7. Identify And Classify 5 Applications You Use Daily As Either System Software Or Application Software.**

Here’s A Brief Classification Of 5 Daily-Used Applications:

1. **Operating System (Windows, Macos)** - **System Software**: Manages Hardware And Software Resources.
2. **Web Browser (Chrome, Firefox)** - **Application Software**: Allows Internet Browsing.
3. **Antivirus Software (Norton, Mcafee)** - **System Software**: Protects Against Malware.
4. **Text Editor (Word, Google Docs)** - **Application Software**: Creates And Edits Documents.
5. **File Management System (Windows Explorer, Finder)** - **System Software**: Manages Files And Directories.

**8. Design A Basic Three-Tier Software Architecture Diagram For A Web Application.**

****

**1. Presentation Tier (Client Tier)**

The User Interface And User Interaction Are Under The Control Of The Presentation Tier. It Functions As The Application’s Front End, Where Users Enter Information And See The Outcomes.

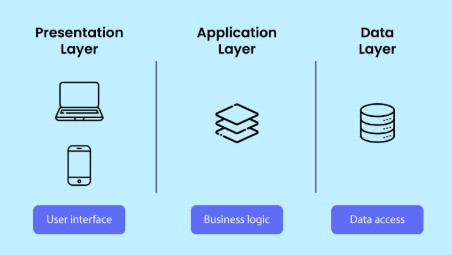
**2. Application Tier (Business Logic Tier)**

The Application Tier Is Where The Core Business Logic Resides. It Processes User Requests, Performs Calculations, Enforces Business Rules, And Interacts With The Data Tier To Retrieve Or Store Data.

**3. Data Tier (Database Tier)**

The Data Tier Is Responsible For Data Management And Storage. It Handles All Database Operations, Including Data Retrieval, Updates, And Management.

**9. Create A Case Study On The Functionality Of The Presentation, Business Logic, And Data Access Layers Of A Given Software System.**

****

**1. Presentation Tier (Client Tier)**

The User Interface And User Interaction Are Under The Control Of The Presentation Tier. It Functions As The Application’s Front End, Where Users Enter Information And See The Outcomes.

**2. Application Tier (Business Logic Tier)**

The Application Tier Is Where The Core Business Logic Resides. It Processes User Requests, Performs Calculations, Enforces Business Rules, And Interacts With The Data Tier To Retrieve Or Store Data.

**3. Data Tier (Database Tier)**

The Data Tier Is Responsible For Data Management And Storage. It Handles All Database Operations, Including Data Retrieval, Updates, And Management.

**10. Explore Different Types Of Software Environments (Development, Testing, Production). Set Up A Basic Environment In A Virtual Machine.**

To Explore Different Software Environments In Java, Let's Look At **Development**, **Testing**, And **Production**:

1. **Development Environment**: Where Java Code Is Written And Locally Tested.
   * Tools: IDE (E.G., Intellij IDEA, Eclipse), JDK (Java Development Kit), Maven/Gradle For Build Automation.
   * Example: A Local Java Project Setup With JDK And Dependencies.
2. **Testing Environment**: A Replica Of The Production Environment Where You Run Unit Tests, Integration Tests, Etc.
   * Tools: Junit, Testng, Or Selenium For Automated Testing.
   * Example: A Separate Testing Environment With Mock Services Or Docker Containers.
3. **Production Environment**: The Live System Where The Java Application Is Deployed For End-Users.
   * Tools: Application Servers (E.G., Tomcat, Wildfly), Monitoring Tools.
   * Example: A Java App Running On A Cloud Server Or On-Premise Server With Proper Security And Scalability.

**Setting Up A Basic Java Environment In A Virtual Machine:**

1. **Install Virtual Machine Software** (E.G., Virtualbox).
2. **Create And Configure VM** (E.G., Ubuntu Or Centos).
3. **Install Java Development Kit (JDK)**:
   * Sudo Apt Update
   * Sudo Apt Install Openjdk-11-Jdk (For Ubuntu)
4. **Install Build Tools** (Maven/Gradle):
   * Sudo Apt Install Maven (For Maven)
5. **Install IDE** (Optional):
   * Download And Set Up Intellij IDEA Or Eclipse.

This Setup Can Be Adapted For Development, Testing, And Production Stages By Configuring Different Tools, Environments, And Workflows For Each Phase.

**11. Write And Upload Your First Source Code File To Github.**

1. **Create A Github Repository**: Go To Github, Click **+** > **New Repository** > Name And Create.
2. **Initialize Git**: In Your Terminal, Run Git Init.
3. **Add Remote**: Git Remote Add Origin Https://Github.Com/Your-Username/Your-Repo.Git.
4. **Stage Changes**: Git Add .
5. **Commit Changes**: Git Commit -M "Your Commit Message"
6. **Push To Github**: Git Push Origin Master

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

### ****GitHub Commit & Push Guide****

1. **Create a GitHub Repo**
   * Go to [GitHub](https://github.com/) → Click **New Repository** → Name it → Click **Create**
2. **Clone Repo**
3. git clone https://github.com/YOUR\_USERNAME/your-repo.git
4. cd your-repo
5. **Add & Commit Changes**
6. git add .
7. git commit -m "Your commit message"
8. **Push Changes**
9. git push origin main
10. **Pull Latest Changes (if needed)**
11. git pull origin main

**13. Create A Student Account On Github And Collaborate On A Small Project With A Classmate.**

1. **Create A Student Github Account**:
   * Go To [Github Education](https://education.github.com/pack).
   * Click **Get Your Github Student Developer Pack**.
   * Sign Up Using Your School Email And Verify.
2. **Create A Repository**:
   * After Logging In, Click **+** > **New Repository**.
   * Name It, Add Description, And Click **Create Repository**.
3. **Invite A Classmate To Collaborate**:
   * In Your Repo, Click **Settings** > **Manage Access** > **Invite A Collaborator**.
   * Enter Your Classmate's Github Username And Click **Add**.
4. **Clone Repository**:
   * Your Classmate Clones The Repo: Git Clone Https://Github.Com/Your-Username/Repo-Name.Git.
5. **Make Changes And Push**:
   * Both Can **Commit** And **Push** Changes:  
     Git Add ., Git Commit -M "Message", Git Push.
6. **Pull Request** (If Working On Separate Branches):
   * When Done, Create A **Pull Request** For Reviewing Changes.

**14. Create A List Of Software You Use Regularly And Classify Them Into The Following Categories: System, Application, And Utility Software.**

**System Software:**

1. Operating System: Windows, Macos
2. Virtual Machine: Virtualbox, Vmware

**Application Software:**

1. Browsers: Chrome, Firefox
2. Office Suites: Microsoft Office, Google Docs
3. Ides: VS Code, Intellij IDEA
4. Media Players: VLC, Windows Media Player

**Utility Software:**

1. File Compression: Winrar, 7-Zip
2. Backup: Google Drive, Dropbox
3. Antivirus: Windows Defender, Avast
4. Disk Cleanup: Ccleaner

**15. Follow A GIT Tutorial To Practice Cloning, Branching, And Merging Repositories.**

**1. Clone A Repository**

* Clone A Repo From Github:
* Git Clone Https://Github.Com/Your-Username/Your-Repo.Git

**2. Create A New Branch**

* Create And Switch To A New Branch:
* Git Checkout -B New-Branch

**3. Make Changes And Commit**

* Edit Files, Then Stage And Commit Changes:
* Git Add .
* Git Commit -M "Made Changes"

**4. Push The Branch To Github**

* Push The Branch To Github:
* Git Push Origin New-Branch

**5. Create A Pull Request**

* Go To Github And Create A PR To Merge New-Branch Into Main.

**6. Merge The Branch**

* After The PR Is Merged, Pull The Changes To Your Local Repo:
* Git Checkout Main
* Git Pull Origin Main

**16. Write A Report On The Various Types Of Application Software And How They Improve Productivity.**

**Report On Types Of Application Software And Their Impact On Productivity**

**1. Word Processing Software**

* **Examples**: Microsoft Word, Google Docs
* **Impact**: These Tools Improve Productivity By Allowing Users To Create, Edit, And Format Documents Quickly. Features Like Templates, Spell Check, And Collaboration Tools Help Streamline Writing Tasks.

**2. Spreadsheet Software**

* **Examples**: Microsoft Excel, Google Sheets
* **Impact**: Spreadsheets Enable Data Analysis, Calculations, And Visualization. They Improve Productivity By Automating Repetitive Tasks, Performing Complex Computations, And Visualizing Data Trends.

**3. Presentation Software**

* **Examples**: Microsoft Powerpoint, Google Slides
* **Impact**: These Tools Help Users Create Visually Engaging Presentations. They Improve Productivity By Providing Templates, Multimedia Integration, And Collaborative Features For Quick Assembly And Editing.

**4. Database Management Software**

* **Examples**: Microsoft Access, Oracle
* **Impact**: Database Software Improves Productivity By Organizing, Managing, And Querying Large Volumes Of Data. It Allows Users To Retrieve, Modify, And Analyze Data Efficiently, Supporting Informed Decision-Making.

**5. Email And Communication Software**

* **Examples**: Microsoft Outlook, Slack, Zoom
* **Impact**: Email And Messaging Software Enhance Productivity By Enabling Efficient Communication, Scheduling, And Collaboration In Real-Time, Reducing Delays And Improving Workflow Coordination.

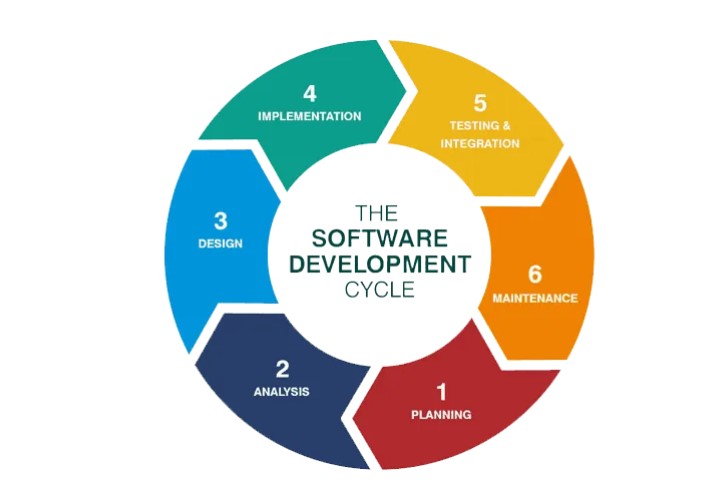
**6. Project Management Software**

* **Examples**: Trello, Asana, Microsoft Project
* **Impact**: These Tools Improve Productivity By Organizing Tasks, Tracking Progress, And Managing Deadlines. They Help Teams Collaborate, Allocate Resources, And Stay On Schedule.

**7. Graphic Design Software**

* **Examples**: Adobe Photoshop, Canva
* **Impact**: Design Software Speeds Up The Creation Of Visuals For Marketing, Social Media, And Web Development. It Provides Tools For Quick Editing, Rendering, And Collaboration.

**17. Create A Flowchart Representing The Software Development Life Cycle (SDLC).**



**18. Write A Requirement Specification For A Simple Library Management System.**

**Requirement Specification For Library Management System**

**1. User Management**

* **Registration**: Users Can Create Accounts With Basic Details (Name, Email, Phone).
* **Login/Logout**: Users Can Securely Log In And Log Out.
* **Profile Management**: Users Can Update Their Profile Information.

**2. Book Management**

* **Add/Edit/Delete Books**: Admin Can Manage Book Details (Title, Author, Genre, Availability).
* **Book Search**: Users Can Search Books By Title, Author, Or Genre.

**3. Transaction Management**

* **Checkout/Return**: Users Can Borrow And Return Books.
* **Late Fees**: Calculate Late Fees Based On Overdue Days.

**4. Notifications**

* **Due Date Reminders**: Notify Users About Upcoming Due Dates.
* **Overdue Alerts**: Alert Users When A Book Is Overdue.

**5. Reporting**

* **Book Status**: Admin Can View Available, Checked-Out, And Overdue Books.
* **User History**: Admin Can View User Transaction History.

**6. Non-Functional Requirements**

* **Usability**: User-Friendly Interface.
* **Security**: Secure User Authentication And Data Protection.
* **Performance**: Fast Book Search And Smooth User Experience.
* **Backup**: Daily Data Backup And Recovery.

**7. System Requirements**

* **Platform**: Web-Based With Mobile Support.
* **Database**: Relational Database (E.G., Mysql).
* **Technologies**: Frontend (HTML, CSS, Javascript), Backend (Node.Js, Django).

**19. Perform A Functional Analysis For An Online Shopping System.**

**Functional Analysis For An Online Shopping System**

**1. User Management**

* **Account Creation/Registration**: Users Can Create Accounts With Basic Details (Name, Email, Password).
* **Login/Logout**: Users Can Securely Log In And Log Out.
* **Profile Management**: Users Can Update Their Personal Details And View Order History.

**2. Product Management**

* **Product Listing**: Admin Can Add, Edit, Or Remove Products (Name, Description, Price, Stock).
* **Product Search**: Users Can Search Products By Category, Price Range, Or Keywords.
* **Product Details**: Users Can View Detailed Information For Each Product (Images, Specs, Reviews).

**3. Shopping Cart**

* **Add/Remove Items**: Users Can Add/Remove Items To/From Their Shopping Cart.
* **View Cart**: Users Can View The Cart With Product Details And Total Price.
* **Update Quantities**: Users Can Modify Quantities Of Items In The Cart.

**4. Checkout And Payment**

* **Order Summary**: Users Can Review Their Order Before Checkout.
* **Payment Options**: Users Can Choose Payment Methods (Credit Card, Paypal, Etc.).
* **Order Confirmation**: System Confirms The Order And Generates An Order Number.

**5. Order Management**

* **Track Order**: Users Can Track The Status Of Their Orders (Shipped, Delivered, Etc.).
* **Cancel/Return**: Users Can Request Order Cancellation Or Return Items As Per Policy.
* **Order History**: Users Can View Past Orders.

**6. Customer Support**

* **Contact Support**: Users Can Contact Customer Support Via Chat Or Email.
* **Faqs**: Provide A List Of Frequently Asked Questions For Quick Support.

**7. Notifications**

* **Order Updates**: Notify Users Of Order Status Changes (Shipped, Delivered).
* **Promotions**: Notify Users Of Sales, Discounts, Or New Products.

**8. Admin Functions**

* **Sales Reporting**: Admin Can Generate Reports On Sales, Revenue, And Inventory.
* **Manage Users**: Admin Can Manage Customer Accounts, Block Or Delete Users If Needed.

**20. Design A Basic System Architecture For A Food Delivery App.**

1. Check if the calculator is a normal calculator or a scientific calculator.
2. Verify that all the buttons are present and text written on them is readable.
3. Check the arithmetic operations are working fine- +, -, /, \*, etc.
4. Verify that BODMAS is applied in case of complex queries and that the correct result is returned.
5. Verify that the calculator gives the correct result in the case of operations containing decimal numbers.
6. Check if the calculator is battery-operated or works on solar power.
7. Verify the outer body material of the calculator.
8. Verify the spacing between the two buttons, the buttons should not be too closely placed.
9. Check the pressure required to press a button, the pressure required should not be too high.
10. Verify the number of digits allowed to enter in the calculator for any operation.
11. Verify the limit of the response value.
12. Verify the functioning of memory functions.
13. Check if the calculator allows navigating through previous calculations.
14. Verify that hitting ‘C’ cancels any digits or operations added.
15. Verify the working of the ON-OFF button in the calculator.

**21. Develop Test Cases For A Simple Calculator Program.**

**Test Cases For Simple Calculator Program**

**1. Addition**

* **Test Case ID**: TC\_Add\_01
* **Input**: 5 + 3
* **Expected Output**: 8

**2. Subtraction**

* **Test Case ID**: TC\_Sub\_01
* **Input**: 10 - 4
* **Expected Output**: 6

**3. Multiplication**

* **Test Case ID**: TC\_Mul\_01
* **Input**: 6 \* 7
* **Expected Output**: 42

**4. Division**

* **Test Case ID**: TC\_Div\_01
* **Input**: 8 / 4
* **Expected Output**: 2

**5. Division By Zero**

* **Test Case ID**: TC\_Div\_02
* **Input**: 10 / 0
* **Expected Output**: Error: Cannot Divide By Zero

**6. Negative Addition**

* **Test Case ID**: TC\_Add\_02
* **Input**: -5 + 3
* **Expected Output**: -2

**7. Decimal Multiplication**

* **Test Case ID**: TC\_Mul\_02
* **Input**: 2.5 \* 4
* **Expected Output**: 10

**8. Invalid Input**

* **Test Case ID**: TC\_Invalid\_01
* **Input**: A + B
* **Expected Output**: Error: Invalid Input

**9. Large Number Addition**

* **Test Case ID**: TC\_Add\_03
* **Input**: 100000 + 200000
* **Expected Output**: 300000

**10. Zero Multiplication**

* **Test Case ID**: TC\_Mul\_03
* **Input**: 0 \* 8
* **Expected Output**: 0

**22. Document A Real-World Case Where A Software Application Required.**

**Real-World Case: Software Application For Online Food Delivery**

**Background**

A Food Delivery Service Faced Challenges With Order Management, Customer Tracking, And Delivery Logistics. The Manual System Was Inefficient And Led To Errors, Delayed Deliveries, And Customer Dissatisfaction.

**Problem**

* **Order Mistakes**: Orders Were Sometimes Mixed Up Or Not Recorded Correctly.
* **Delivery Delays**: Drivers Had Trouble Navigating And Tracking Real-Time Orders.
* **Lack Of Real-Time Updates**: Customers Didn’t Have Visibility Into The Status Of Their Orders.

**Solution**

The Company Developed A **Custom Online Food Delivery App** With The Following Features:

1. **Order Management**: Customers Could Easily Place Orders Through The App.
2. **Real-Time Tracking**: Drivers And Customers Could Track Orders In Real Time.
3. **Notifications**: Automated Notifications For Order Confirmation, Delivery Updates, And Arrival Times.

**Outcome**

* **Improved Efficiency**: Orders Were Processed Accurately And Quickly.
* **Better Customer Experience**: Customers Received Timely Updates And Could Track Their Orders.
* **Increased Sales**: The Streamlined Process Led To Faster Deliveries And Higher Customer Satisfaction.

**23. Create A DFD For A Hospital Management System.**

Here's a simplified version of the **Data Flow Diagram (DFD)** for a **Hospital Management System**:

**Level 0: Context Diagram**

**External Entities:**

1. **Patient**
2. **Doctor**
3. **Admin**
4. **Insurance Provider**

**System:**

* **Hospital Management System (HMS)**

**Data Flows:**

* **Patient → HMS**: Patient details, appointment requests.
* **HMS → Patient**: Appointment confirmations, medical records.
* **Doctor → HMS**: Treatment updates, medical records.
* **HMS → Doctor**: Patient details, appointments.
* **Admin → HMS**: Registration, management tasks.
* **HMS → Admin**: Appointment, billing management.
* **Insurance Provider → HMS**: Insurance details, claims.
* **HMS → Insurance Provider**: Claims processing, billing.

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

| Hospital Management System |

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

| |

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

| | | |

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

| Patient | | Doctor | | Admin | | Insurance |

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

**Level 1: Decomposed DFD**

**Processes:**

1. **Manage Patient Information**: Handles registration and updates.
2. **Manage Appointments**: Manages patient appointment scheduling.
3. **Medical Record Management**: Manages treatment records.
4. **Billing and Payments**: Manages payments and billing.
5. **Insurance Processing**: Handles insurance claims.

**Data Stores:**

* **Patient Database**: Stores patient info.
* **Appointment Database**: Stores appointment data.
* **Medical Record Database**: Stores medical records.
* **Billing Database**: Stores billing information.

**Example Data Flow:**

* **Patient → Manage Patient Information**: Registration info.
* **Manage Appointments → Appointment Database**: Store appointment data.
* **Doctor → Medical Record Management**: Treatment details.
* **Insurance Processing → Insurance Provider**: Claims information.

**24. Build A Simple Desktop Calculator Application Using A GUI Library.**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class Calculator implements ActionListener {

private JFrame frame;

private JTextField textField;

private JButton[] numberButtons = new JButton[10];

private JButton[] functionButtons = new JButton[6];

private String[] functionNames = {"+", "-", "\*", "/", "=", "C"};

private double num1, num2, result;

private char operator;

public Calculator() {

frame = new JFrame("Calculator");

frame.setSize(400, 500);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setLayout(new BorderLayout());

textField = new JTextField();

textField.setFont(new Font("Arial", Font.BOLD, 24));

textField.setHorizontalAlignment(JTextField.RIGHT);

frame.add(textField, BorderLayout.NORTH);

JPanel buttonPanel = new JPanel();

buttonPanel.setLayout(new GridLayout(4, 4, 5, 5));

// Creating number buttons (0-9)

for (int i = 0; i < 10; i++) {

numberButtons[i] = new JButton(String.valueOf(i));

numberButtons[i].setFont(new Font("Arial", Font.BOLD, 20));

numberButtons[i].addActionListener(this);

}

// Creating function buttons (+, -, \*, /, =, C)

for (int i = 0; i < 6; i++) {

functionButtons[i] = new JButton(functionNames[i]);

functionButtons[i].setFont(new Font("Arial", Font.BOLD, 20));

functionButtons[i].addActionListener(this);

}

// Adding buttons to the panel in a 4x4 layout

buttonPanel.add(numberButtons[7]);

buttonPanel.add(numberButtons[8]);

buttonPanel.add(numberButtons[9]);

buttonPanel.add(functionButtons[3]); // "/"

buttonPanel.add(numberButtons[4]);

buttonPanel.add(numberButtons[5]);

buttonPanel.add(numberButtons[6]);

buttonPanel.add(functionButtons[2]); // "\*"

buttonPanel.add(numberButtons[1]);

buttonPanel.add(numberButtons[2]);

buttonPanel.add(numberButtons[3]);

buttonPanel.add(functionButtons[1]); // "-"

buttonPanel.add(numberButtons[0]);

buttonPanel.add(functionButtons[5]); // "C"

buttonPanel.add(functionButtons[4]); // "="

buttonPanel.add(functionButtons[0]); // "+"

frame.add(buttonPanel, BorderLayout.CENTER);

frame.setVisible(true);

}

@Override

public void actionPerformed(ActionEvent e) {

for (int i = 0; i < 10; i++) {

if (e.getSource() == numberButtons[i]) {

textField.setText(textField.getText() + i);

}

}

for (int i = 0; i < 4; i++) { // Handling operations (+, -, \*, /)

if (e.getSource() == functionButtons[i]) {

num1 = Double.parseDouble(textField.getText());

operator = functionNames[i].charAt(0);

textField.setText("");

}

}

if (e.getSource() == functionButtons[4]) { // "=" button

num2 = Double.parseDouble(textField.getText());

switch (operator) {

case '+': result = num1 + num2; break;

case '-': result = num1 - num2; break;

case '\*': result = num1 \* num2; break;

case '/': result = num1 / num2; break;

}

textField.setText(String.valueOf(result));

}

if (e.getSource() == functionButtons[5]) { // "C" button

textField.setText("");

}

}

public static void main(String[] args) {

new Calculator();

}

}

**25. Draw A Flowchart Representing The Logic Of A Basic Online Registration System.**

