**Module 1 – Overview of IT Industry (Lab Exercise)**

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

**C** :

#include <stdio.h>

int main() {

printf("Hello World");

return 0;

}

**Python** :

print("Hello World ")

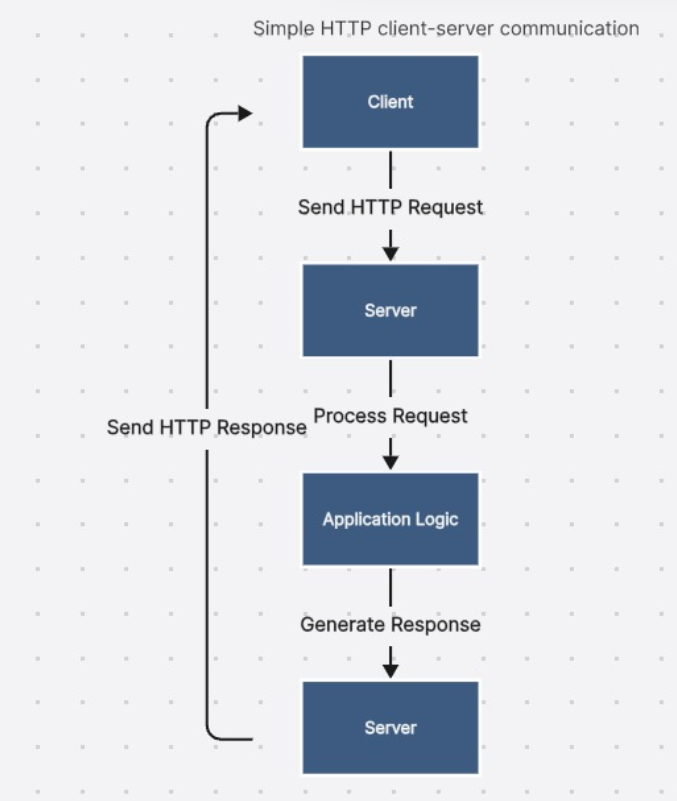
**Comparison:**

**Syntax Simplicity:** Python is simpler and more readable, needing just one line.

**Structure:** C requires more structure like #include, main() function, and return statement.

**Use Case:** Python is interpreted and best for beginners; C is compiled and better for system-level programming.

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



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

* To simulate an HTTP GET request and display the content of a web page:

**bash**

curl https://example.com/

This command will retrieve the HTML content of the specified URL and display it in the terminal.

* To simulate an FTP download, you can use the command:

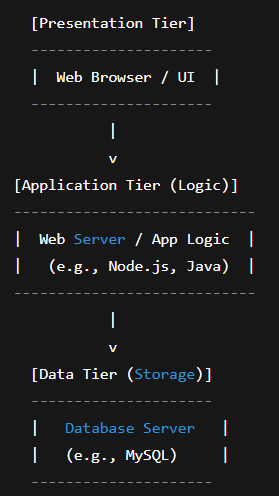
**bash**

curl -O ftp://ftp.example.com/file.zip

This command downloads the file file.zip from the FTP server and saves it to the local directory with the same name, [according to Oxylabs](https://oxylabs.io/blog/curl-download-file).

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

* **Presentation Tier**: User interface (browser or mobile app)
* **Application Tier**: Handles business logic
* **Data Tier**: Manages data storage and retrieval



1. Write and upload your first source code file to GitHub.

Step 1: Create a Simple Source Code File

Step 2: Create a New Git Repository

Step 3: Create a Repository on GitHub

Step 4: Connect Local Repo to GitHub

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

**Step 1: Create a GitHub Repository**

* + Go to <https://github.com/new>
  + Enter a repository name (e.g., my-first-project)
  + Choose **Public** or **Private**
  + **DO NOT** check “Initialize with README” (important for command line setup)
  + Click **Create repository**

**Step 2: Set Up Locally and Push Code**

#### A. Open Terminal/Command Prompt

bash

CopyEdit

mkdir my-first-project

cd my-first-project

#### B. Create a Simple File

bash

CopyEdit

echo "print('Hello GitHub')" > hello.py

#### C. Initialize Git & Commit Code

bash

CopyEdit

git init # Initialize Git repo

git add hello.py # Stage the file

git commit -m "Initial commit" # Commit with a message

#### D. Connect to GitHub Repo & Push

bash

CopyEdit

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

git branch -M main

git push -u origin main

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

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

* + Visit <https://github.com/join>
  + Fill in your **username**, **email**, and **password**, then create your account.
  + After verification, apply for **GitHub Student Developer Pack**:  
     <https://education.github.com/pack>
    - * + Use your **college/university email** (if available)
        + Upload proof of student status (ID card, fee receipt, etc.)

**Step 2: Collaborate on a Project**

#### One Person Creates the Repo

* + Go to [GitHub.com](https://github.com), click **+** → **New repository**
  + Name it something like mini-project
  + Choose **Private** (if only for classmates) or **Public**
  + Initialize with a README.md if you like

#### Add Collaborator

* + Go to your repo → **Settings** → **Collaborators**
  + Enter your classmate’s GitHub **username**
  + Click **Add collaborator** → they’ll receive an email invitation

**Step 3: Work on the Project Together**

Both collaborators can now:

bash

CopyEdit

# Clone the repo

git clone https://github.com/username/mini-project.git

# Add or change files

echo "print('Collaborating!')" > hello.py

# Commit and push changes

git add .

git commit -m "Added hello.py"

git push

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

* Operating System: Windows, macOS, Linux, Android, iOS
  + - * Application software
* Productivity: Microsoft Word/Excel/PowerPoint, Google Docs/Sheets/Slides, Notion, email clients
* Browsers: Google Chrome, Mozilla Firefox, Microsoft Edge, Safari
* Communication: Zoom, Slack, Microsoft Teams, WhatsApp, Skype
* Multimedia: VLC Media Player, Spotify, YouTube, Netflix
* Image/Video Editing: Adobe Photoshop, Canva, CapCut
  + - * Utility software
* System Maintenance/Optimization: Disk Cleanup, CCleaner, Norton Utilities, Windows Disk Defragmenter
* Security: Antivirus software (e.g., Norton, Avast, McAfee)
* File Management: File Explorer (Windows), Finder (macOS), WinRAR (file compression)

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

**Step 1: Create a Repository on GitHub**

* + Go to <https://github.com/new>
  + Name it: git-practice
  + Check **"Add a README file"**
  + Click **Create repository**

**Step 2: Clone the Repository**

bash

CopyEdit

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

cd git-practice

**Step 3: Create and Switch to a New Branch**

bash

CopyEdit

git checkout -b feature-branch

**You’re now on a new branch called feature-branch**.

**Step 4: Make a Change**

bash

CopyEdit

echo "This is a test feature file." > feature.txt

git add feature.txt

git commit -m "Add feature.txt"

**Step 5: Merge the Branch Back to Main**

bash

CopyEdit

git checkout main

git merge feature-branch

**If there are no conflicts, it merges cleanly**.

**Step 6: Push Changes to GitHub**

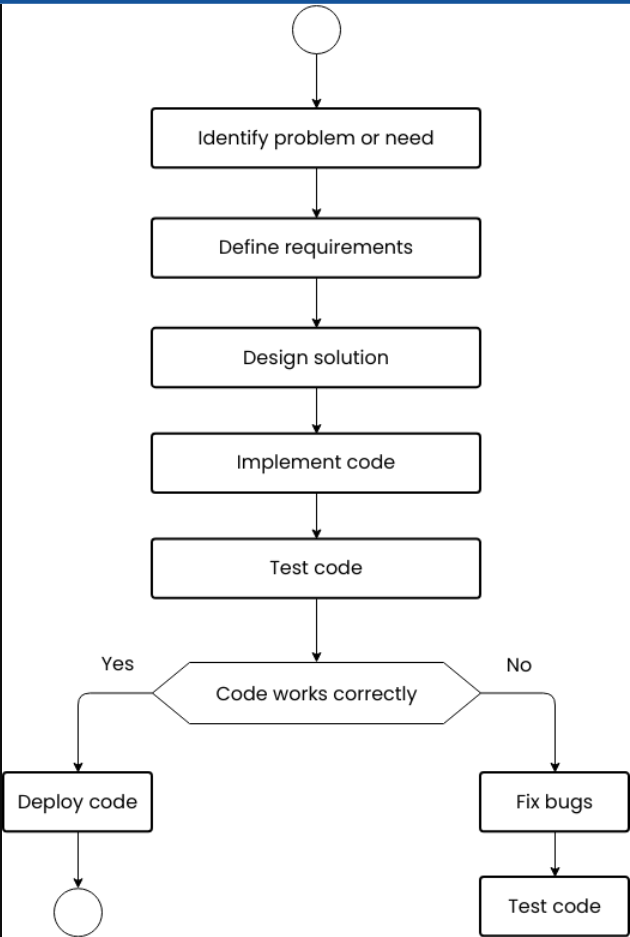
bash

CopyEdit

git push origin main

**Now check your GitHub repo to see the merged changes!**

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

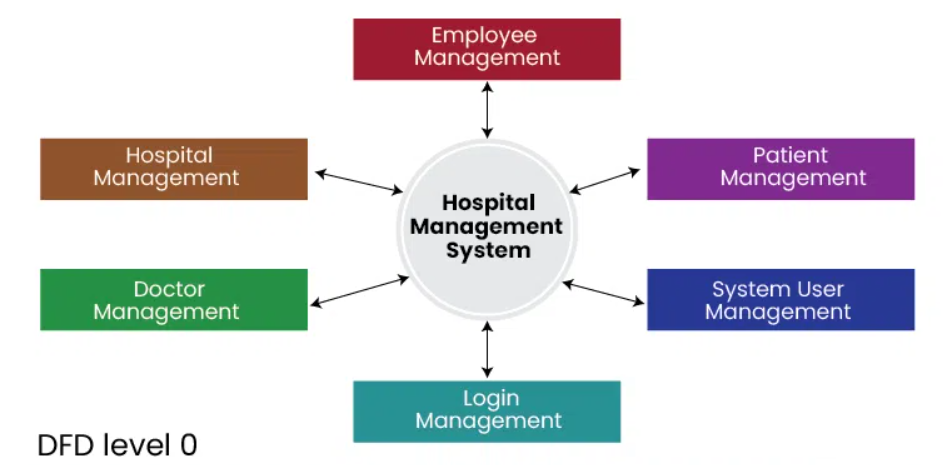


1. **Develop test cases for a simple calculator program.**

**Test Cases – Simple Calculator Program**

| **Test Case ID** | **Description** | **Input** | **Expected Output** | **Remarks** |
| --- | --- | --- | --- | --- |
| TC001 | Add two positive numbers | 5 + 3 | 8 | Basic addition |
| TC002 | Add positive and negative number | 10 + (-4) | 6 | Addition with negative |
| TC003 | Subtract two numbers | 9 - 4 | 5 | Basic subtraction |
| TC004 | Subtract larger from smaller | 3 - 5 | -2 | Negative result |
| TC005 | Multiply two positive numbers | 4 × 3 | 12 | Basic multiplication |
| TC006 | Multiply with zero | 6 × 0 | 0 | Zero multiplication |
| TC007 | Divide two numbers | 8 ÷ 2 | 4 | Basic division |
| TC008 | Divide by zero | 7 ÷ 0 | Error / Exception | Division by zero check |
| TC009 | Divide resulting in decimal | 7 ÷ 2 | 3.5 | Floating-point result |
| TC010 | Add two decimal numbers | 2.5 + 1.5 | 4.0 | Floating-point addition |
| TC011 | Input non-numeric values | "a" + 2 | Error / Invalid Input | Input validation |
| TC012 | Chained operations (if supported) | 2 + 3 × 4 | 14 | Operator precedence |
| TC013 | Negative result in division | -8 ÷ 2 | -4 | Negative division |
| TC014 | Multiply negative and positive | -3 × 6 | -18 | Sign handling |
| TC015 | Very large numbers | 1,000,000 × 1,000,000 | 1,000,000,000,000 | Performance test |

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



1. **Build a simple desktop calculator application using a GUI library.**

from tkinter import \*

expression = ""

def press(num):

"""

Updates the global expression string with the pressed number or operator.

"""

global expression

expression = expression + str(num)

equation.set(expression)

def equalpress():

"""

Evaluates the expression and displays the result.

Handles errors like division by zero or invalid syntax.

"""

try:

global expression

total = str(eval(expression))

equation.set(total)

expression = ""

except:

equation.set(" error ")

expression = ""

def clear():

"""

Clears the expression and the display.

"""

global expression

expression = ""

equation.set("")

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

root = Tk()

root.title("Simple Calculator")

root.geometry("310x200")

root.resizable(False, False) # Prevent resizing for a fixed layout

equation = StringVar()

expression\_field = Entry(root, textvariable=equation, font=('Arial', 16), bd=5, insertwidth=4, width=18, justify='right')

expression\_field.grid(row=0, column=0, columnspan=4, padx=5, pady=5)

# Define button layout

buttons = [

('7', 1, 0), ('8', 1, 1), ('9', 1, 2), ('/', 1, 3),

('4', 2, 0), ('5', 2, 1), ('6', 2, 2), ('\*', 2, 3),

('1', 3, 0), ('2', 3, 1), ('3', 3, 2), ('-', 3, 3),

('0', 4, 0), ('.', 4, 1), ('=', 4, 2), ('+', 4, 3)

]

# Create and place buttons

for (text, row, col) in buttons:

if text == '=':

Button(root, text=text, fg='black', bg='lightgray',

command=equalpress, height=2, width=8).grid(row=row, column=col, sticky="nsew", padx=2, pady=2)

else:

Button(root, text=text, fg='black', bg='lightgray',

command=lambda t=text: press(t), height=2, width=8).grid(row=row, column=col, sticky="nsew", padx=2, pady=2)

clear\_button = Button(root, text='Clear', fg='black', bg='lightgray',

command=clear, height=2, width=8)

clear\_button.grid(row=5, column=0, columnspan=4, sticky="nsew", padx=2, pady=2)

# Configure column and row weights for responsive layout (within fixed size)

for i in range(4):

root.grid\_columnconfigure(i, weight=1)

for i in range(6): # 5 rows for buttons + 1 for entry

root.grid\_rowconfigure(i, weight=1)

root.mainloop()

**Explanation:**

* Import **tkinter**: The necessary GUI library is imported.
* **expression** variable: A global string variable **expression** stores the mathematical expression as it's being built.
* press(num) function: Appends the clicked number or operator to the expression and updates the display.
* equalpress() function: Evaluates the expression using eval() and displays the result. It includes error handling for invalid input or division by zero.
* clear() function: Resets the expression and clears the display.
* **Main Window Setup:**
* Tk() creates the main window.
* title() sets the window title.
* geometry() defines the window size.
* resizable(False, False) prevents the user from resizing the window.
* **Display Field:** An Entry widget is used to display the input and results, linked to the equation StringVar.
* **Buttons:**
* A list buttons defines the text and grid position for each button.
* A loop iterates through this list to create Button widgets.
* command attribute links buttons to press() or equalpress() functions, using lambda for passing arguments to press().
* **Clear Button:** A dedicated "Clear" button is created and linked to the clear() function.
* **Layout Management:** grid() is used to arrange widgets in a grid layout. sticky="nsew" makes buttons expand to fill their grid cells. grid\_columnconfigure and grid\_rowconfigure are used to distribute space evenly.
* mainloop(): Starts the Tkinter event loop, which keeps the window open and responsive to user interactions.

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

