# Important:

* Only use VS code for the assignments during practical. Eclipse and PyCharm are not properly setup.
* For Java assignments openjdk-8 is used
* Follow the steps to run and test the code in this document for each assignment. Same will work on lab computers too.
* For Assignment -7 install the necessary packages : Flask and requests

# 

# Assignment -1

Write this code in VS code only. Eclipse is not set up on the systems

1. Create three folders
   * 1. client
     2. remotes
     3. server
2. In the remotes folder, create two files Search.java and SearchQuery.java
3. Write following code in **Search.java**

**Search.java**

package **remotes**;

import **java.rmi.Remote**;

import **java.rmi.RemoteException**;

**public** **interface** Search **extends** Remote {

**public** **String** query(**String** search) **throws** **RemoteException**;

}

1. Write following code in **SearchQuery.java**

**SearchQuery.java**

**package remotes;**

**import java.rmi.\*;**

**import java.rmi.server.\*;**

**public class SearchQuery extends UnicastRemoteObject implements Search {**

**public SearchQuery() throws RemoteException {**

**super();**

**}**

**public String query(String search) throws RemoteException {**

**String result = "No results found";**

**if (search.equals("p2p")) {**

**result = "Found 1 result";**

**}**

**return result;**

**}**

**}**

1. In the server folder, create a file named **SearchServer.java** and write following code in it.

**SearchServer.java**

package **server**;

import **java.rmi.\***;

import **java.rmi.registry.\***;

import **remotes.Search**;

import **remotes.SearchQuery**;

**public** **class** SearchServer {

**public** **static** **void** main(**String**[] args) {

try {

**Search** search = new SearchQuery();

**Registry** registry = LocateRegistry.createRegistry(1099);

Naming.rebind("rmi://localhost:1099"+ "/REMOTE\_SEARCH", search);

System.out.println("Search Server ready");

} catch (**Exception** e) {

System.out.println("Search Server main " + e.getMessage());

}

}

}

1. In the client folder, create a file named ClientRequest.java and write following code in it.

**ClientRequest.java**

package **client**;

import **java.rmi.\***;

import **remotes.Search**;

**public** **class** ClientRequest {

**public** **static** **void** main(**String**[] args) {

try {

**String** search = (args.length < 1) ? "p2p" : args[0];

**String** url = "rmi://localhost:1099/REMOTE\_SEARCH";

**Search** access = (Search) Naming.lookup(url);

**String** result = access.query(search);

System.out.println("Found: " + result);

} catch (**Exception** e) {

System.out.println("ClientRequest exception: " + e.getMessage());

}

}

}

1. To compile all the java code run the following command in the terminal of VS code.

***javac .\remotes\\*.java***

***javac .\server\\*.java***

***javac .\client\\*.java***

1. To start the server run the following command in the terminal

**java server.SearchServer**

1. Now, open another terminal in VS code and run following command to run the client

**java client.ClientRequest**

# Assignment - 2

Perform this assignment in VS code only. Eclipse is not setup on the lab systems. For testing this code on your local system make sure you have openjdk-8 properly setup.

1. Create a file name calculator.idl, and write following code in it.

module calculator\_module {

interface Calculator {

long add(in long a , in long b);

long subtract(in long a, in long b);

long multiply(in long a, in long b);

long divide(in long a, in long b);

oneway void shutdown();

};

};

1. Run the following command to compile the calculator.idl

***idlj -fall calculator.idl***

1. After running the above a command a folder will be created named calculator\_module.
2. Create two more folders
   1. client
   2. server
3. In the server folder create two files
   1. CalculatorImpl.java
   2. CalculatorServer.java
4. In the client folder create a file named CalculatorClient.java

**CalculatorImpl.java**

**package server;**

**import org.omg.CORBA.ORB;**

**import calculator\_module.CalculatorPOA;**

**public class CalculatorImpl extends CalculatorPOA {**

**private ORB orb;**

**public void setORB(ORB orb\_val) {**

**orb = orb\_val;**

**}**

***// implement add() method***

**@Override**

**public int add(int a, int b) {**

**return a + b;**

**}**

***// implement subtract() method***

**@Override**

**public int subtract(int a, int b) {**

**return a - b;**

**}**

***// implement multiply() method***

**@Override**

**public int multiply(int a, int b) {**

**return a \* b;**

**}**

***// implement divide() method***

**@Override**

**public int divide(int a, int b) {**

**return a / b;**

**}**

***// implement shutdown() method***

**@Override**

**public void shutdown() {**

**orb.shutdown(false);**

**}**

**}**

**CalculatorSever.java**

**package server;**

**import org.omg.CORBA.ORB;**

**import org.omg.CosNaming.\*;**

**import org.omg.PortableServer.\*;**

**import calculator\_module.Calculator;**

**import calculator\_module.CalculatorHelper;**

**public class CalculatorServer {**

**public static void main(String args[]){**

**try {**

***// create and initialize the ORB***

**ORB orb = ORB.init(args, null);**

***// get reference to rootpoa & activate the POAManager***

**POA rootpoa = (POA)orb.resolve\_initial\_references("RootPOA");**

**rootpoa.the\_POAManager().activate();**

***// create servant and register it with the ORB***

**CalculatorImpl calculatorImpl = new CalculatorImpl();**

**calculatorImpl.setORB(orb);**

***// get object reference from the servant***

**org.omg.CORBA.Object ref = rootpoa.servant\_to\_reference(calculatorImpl);**

**Calculator href = CalculatorHelper.narrow(ref);**

***// get the root naming context***

***// NameService invokes the transient name service***

**org.omg.CORBA.Object objRef = orb.resolve\_initial\_references("NameService");**

***// Use NamingContextExt which is part of the Interoperable***

***// Naming Service (INS) specification.***

**NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);**

***// bind the Object Reference in Naming***

**String name = "Calculator";**

**NameComponent path[] = ncRef.to\_name( name );**

**ncRef.rebind(path, href);**

**System.out.println("CalculatorServer ready and waiting ...");**

***// wait for invocations from clients***

**orb.run();**

**} catch (Exception e) {**

**System.err.println("ERROR: " + e);**

**e.printStackTrace(System.out);**

**} finally {**

**System.out.println("CalculatorServer Exiting ...");**

**}**

**}**

**}**

**CalculatorClient.java**

**package client;**

**import org.omg.CORBA.ORB;**

**import org.omg.CORBA.ORBPackage.InvalidName;**

**import org.omg.CosNaming.\*;**

**import calculator\_module.Calculator;**

**import calculator\_module.CalculatorHelper;**

**public class CalculatorClient {**

**public static void main(String args[]) {**

**try {**

***// create and initialize the ORB***

**ORB orb = ORB.init(args, null);**

***// get the root naming context***

***// NameService invokes the transient name service***

**org.omg.CORBA.Object objRef = orb.resolve\_initial\_references("NameService");**

***// Use NamingContextExt which is part of the Interoperable***

***// Naming Service (INS) specification.***

**NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);**

***// resolve the Object Reference in Naming***

**String name = "Calculator";**

**Calculator calculator = CalculatorHelper.narrow(ncRef.resolve\_str(name));**

**System.out.println("Obtained a handle on server object");**

**System.out.println(calculator.add(1, 2));**

**System.out.println(calculator.subtract(1, 2));**

**System.out.println(calculator.multiply(1, 2));**

**System.out.println(calculator.divide(1, 2));**

**} catch (Exception e) {**

**System.out.println("ERROR : " + e);**

**e.printStackTrace(System.out);**

**}**

**}**

**}**

* To run the above code
  + Open a terminal in VS code and compile all the java code :

**javac .\calculator\_module\\*.java**

**javac .\server\\*.java**

**javac .\client\\*.java**

* + Start the orbd server : ***orbd -ORBInitialPort 1050 -ORBInitialHost localhost***
  + Open another terminal in VS code and start the CalculatorServer: ***java server.CalculatorServer -ORBInitialPort 1050 -ORBInitialHost localhost***
  + Open another terminal in VS code and start the CalculatorClient: ***java client.CalculatorClient -ORBInitialPort 1050 -ORBInitialHost localhost***

# Assignment - 3 (OpenMP)

Create a file named **main.c**  and write following code in it.

#include<stdio.h>

#include<omp.h>

#define N 100

#define NUM\_PROCESSORS 4

**int** main()

{

**int** arr[N];

for (**int** i = 0; i < N; i++)

{

arr[i] = sizeof(**int**) \* i;

}

**int** sum = 0;

**int** PARTIAL\_SUM[NUM\_PROCESSORS];

# pragma omp parallel num\_threads(NUM\_PROCESSORS)

{

**int** thread\_id = omp\_get\_thread\_num();

**int** start = thread\_id \* (N / NUM\_PROCESSORS);

**int** end = (thread\_id + 1) \* (N / NUM\_PROCESSORS);

PARTIAL\_SUM[thread\_id] = 0;

for (**int** i = start; i < end; i++)

{

PARTIAL\_SUM[thread\_id] += arr[i];

}

}

for (**int** i = 0; i < NUM\_PROCESSORS; i++)

{

sum += PARTIAL\_SUM[i];

printf("Partial sum of thread %d: %d\n", i, PARTIAL\_SUM[i]);

}

printf("Sum: %d\n", sum);

return 0;

}

To run the above code open a terminal in the VS code and execute the following command:

***gcc -fopenmp main.c -o output***

After run this command run **.\output**

# Assignment - 4 (Clock Synchronization)

* Create Two files
  + server.py
  + client.py

**server.py**

**import socket**

**import time**

**import random**

**import json**

**SERVER\_IP = "127.0.0.1"**

**PORT = 5000**

**def get\_local\_time():**

**return random.randint(int(time.time() - 1e5), int(time.time() + 1e5))**

**def main():**

***## Create server socket***

**server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**server\_socket.bind((SERVER\_IP, PORT))**

**server\_socket.listen(1)**

***## Get local time***

**server\_local\_time = get\_local\_time()**

**print(f"Time server listening on {SERVER\_IP}:{PORT}")**

**print(f"Server time: {server\_local\_time}")**

**is\_client\_enough = False**

**clients = []**

**while not is\_client\_enough:**

***## Accept client connection***

**client\_socket, client\_address = server\_socket.accept()**

**print(f"Connection established with {client\_address}")**

**clients.append(client\_socket)**

**option = input("Do you want to add more clients? (y/n) ")**

**if option == "n" or option == "N":**

**is\_client\_enough = True**

**else:**

**print("Waiting for more clients..." + "\n")**

**client\_local\_times = []**

***## Get local time from all clients***

**for client\_socket in clients:**

**time\_req\_body = json.dumps({"operation": "time\_req"})**

**client\_socket.send(time\_req\_body.encode())**

**client\_local\_time\_response = json.loads(client\_socket.recv(1024).decode())**

**client\_local\_times.append(float(client\_local\_time\_response["client\_time"]))**

***## Calculate adjusted time***

**average\_offset = sum(client\_local\_times) / len(client\_local\_times)**

**adjusted\_time\_offset = (server\_local\_time + average\_offset) / 2**

***## Send adjusted time to all clients***

**for i, client\_socket in enumerate(clients):**

**print(**

**f"Client {client\_socket.getpeername()} LocalTime : {client\_local\_times[i]}"**

**)**

**adjusted\_time = json.dumps(**

**{**

**"adjusted\_time": client\_local\_times[i] - adjusted\_time\_offset,**

**"operation": "time\_adj",**

**}**

**)**

**client\_socket.send(str(adjusted\_time).encode())**

**print(f"Adjusted time sent to {client\_socket.getpeername()}")**

**server\_socket.close()**

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

**main()**

**client.py**

**import socket**

**import time**

**import json**

**import random**

**SERVER\_IP = "127.0.0.1"**

**PORT = 5000**

**def get\_local\_time():**

**return random.randint(int(time.time() - 1e5), int(time.time() + 1e5))**

**def main():**

***## Connect to server***

**client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**client\_socket.connect((SERVER\_IP, PORT))**

**print(f"Connected to {SERVER\_IP}:{PORT}")**

***## Get local time***

**client\_local\_time = get\_local\_time()**

**time\_adjusted = False**

**while not time\_adjusted:**

**server\_res = json.loads(client\_socket.recv(1024).decode())**

**if server\_res["operation"] == "time\_req":**

***## Send local time to server***

**print(f"Local time: {client\_local\_time}")**

**client\_socket.send(json.dumps({"client\_time": client\_local\_time}).encode())**

**if server\_res["operation"] == "time\_adj":**

***## Adjust local time***

**print(f"Time adjustment: {server\_res['adjusted\_time']}")**

**client\_local\_time += float(server\_res["adjusted\_time"])**

**print(f"Adjusted time: {client\_local\_time}")**

**time\_adjusted = True**

**client\_socket.close()**

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

**main()**

**To run the code:**

1. Open a terminal and start the server : **python server.py**
2. Open another terminal and start a client: **python client.py**
3. Follow the prompts on the server to execute the program completely.

# Assignment - 5 (Token Ring)

**server.py**

**import socket**

**import threading**

**TOKEN = "TOKEN"**

**PORT = 8080**

**BUFFER\_SIZE = 1024**

**class TokenRingServer:**

**def \_\_init\_\_(self):**

**self.server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**self.clients = []**

**self.client\_threads = []**

**self.running = False**

**def start(self):**

**self.server\_socket.bind(("localhost", PORT))**

**self.server\_socket.listen()**

**self.running = True**

**print("Server started. Listening for connections...")**

**try:**

**while self.running:**

***## Accept new connections***

**client\_socket, client\_address = self.server\_socket.accept()**

**print(f"New client connected: {client\_address}")**

**self.clients.append(client\_socket)**

***## If this is the first client, send the token***

**if len(self.clients) == 1:**

***# Send the token to the first client***

**client\_socket.send(TOKEN.encode())**

***## Start a new thread to handle the client***

**thread = threading.Thread(**

**target=self.handle\_client, args=(client\_socket,)**

**)**

**thread.start()**

**self.client\_threads.append(thread)**

**except KeyboardInterrupt:**

**self.stop()**

**def handle\_client(self, client\_socket):**

**while self.running:**

***## Receive data from the client***

**data = client\_socket.recv(BUFFER\_SIZE).decode()**

***## select the next client to send the token to***

**next\_client = self.clients[**

**(self.clients.index(client\_socket) + 1) % len(self.clients)**

**]**

***## If the client sends CLOSE, remove it from the list of clients and close the connection***

**if data == "CLOSE":**

**print(f"Client disconnected: {client\_socket.getpeername()}")**

**self.clients.remove(client\_socket)**

**client\_socket.close()**

**data = TOKEN**

**break**

***## If the client sends TOKEN, send it to the next client***

**if data == TOKEN:**

**print("Received token")**

**if len(self.clients) >= 1:**

**if self.running:**

**print("Sending token to next client")**

**next\_client.send(TOKEN.encode())**

**else:**

**print("Server stopped. Not sending token to next client")**

**break**

**def stop(self):**

**self.running = False**

**print("Closing server..")**

***## Send close signal to all clients***

**for client in self.clients:**

**print(f"Sending close signal to {client.getpeername()}")**

**client.send("CLOSE".encode())**

**client.close()**

***## Wait for all threads to finish***

**for thread in self.client\_threads:**

**thread.join()**

**self.server\_socket.close()**

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

**server = TokenRingServer()**

**server.start()**

**client.py**

**import socket**

**SERVER\_ADDRESS = ("localhost", 8080)**

**BUFFER\_SIZE = 1024**

**class TokenRingClient:**

**def \_\_init\_\_(self):**

**self.client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**def connect(self):**

**self.client\_socket.connect(SERVER\_ADDRESS)**

**print("Connected to server")**

**def start(self):**

**try:**

**while True:**

**data = self.client\_socket.recv(BUFFER\_SIZE).decode()**

**if data == "TOKEN":**

**print("Token received. Accessing resource.")**

***# Perform operations on the resource***

***# Simulating work on the resource***

**print("Working on the resource...")**

***# Simulating work by sleeping for 5 seconds***

**import time**

**time.sleep(5)**

**print("Resource access complete. Releasing token.")**

**self.client\_socket.send("TOKEN".encode())**

**if data == "CLOSE":**

**print("Closing client..")**

**self.stop()**

**break**

**except KeyboardInterrupt:**

**print("Closing client..")**

**self.client\_socket.send("CLOSE".encode())**

**self.stop()**

**def stop(self):**

**self.client\_socket.close()**

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

**client = TokenRingClient()**

**client.connect()**

**client.start()**

**To run the above code:**

* Open terminal and start server : **python server.py**
* Start 2-3 clients in other terminals : **python client.py**

# Assignment - 6 (Leader Election )

**bully.py**

**class Bully:**

**def \_\_init\_\_(self, num\_process=5):**

***# Initialize the Bully object with the number of processes and their states***

**self.num\_process = num\_process**

**self.state = [True for \_ in range(num\_process)]**

**self.leader = num\_process**

**def election(self, process\_id):**

***# Perform the election algorithm to elect a coordinator***

**print(f"Process {process\_id} is sending election messages to higher processes")**

**cod = process\_id**

**for i in range(process\_id + 1, self.num\_process + 1):**

**if self.state[i - 1]:**

**print(**

**f"Process {process\_id} is sending election message to process {i}"**

**)**

**cod = i**

**print(f"Process {cod} is sending coordinator message to all")**

***# Update the leader to the elected coordinator***

**self.leader = cod**

**print(f"Process {self.leader} is now coordinator.")**

**def up(self, process\_id):**

***# Bring up a process and trigger an election if necessary***

**if self.state[process\_id - 1]:**

**print(f"Process {process\_id} is already up")**

**return**

**else:**

**self.state[process\_id - 1] = True**

**print(f"Process {process\_id} is up")**

**self.election(process\_id)**

**def down(self, process\_id):**

***# Bring down a process and initiate a new election if the leader is down***

**if not self.state[process\_id - 1]:**

**print(f"Process {process\_id} is already down.")**

**else:**

**self.state[process\_id - 1] = False**

**print(f"Process {process\_id} is now down")**

**if self.leader == process\_id:**

***# If the leader is down, randomly select a new active process and trigger an election***

**active = [i for i, \_ in enumerate(self.state) if i]**

**import random**

**index = random.randint(0, len(active) - 1)**

**self.election(active[index])**

**def message(self, process\_id):**

***# Send a message and check if the coordinator is active***

**if self.state[process\_id - 1]:**

**if self.state[self.leader - 1]:**

**print("OK")**

**else:**

***# If the coordinator is down, initiate a new election***

**self.election(process\_id)**

**else:**

**print(f"Process {process\_id} is down.")**

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

***# Create a Bully object***

**bully = Bully()**

**print("5 Active processes are:")**

**print("Processes up = p1 p2 p3 p4 p5")**

**print(f"Process {bully.leader} is the coordinator")**

**choice = 5**

**while choice != 4:**

**print("-------------------------------------")**

**print("1) Up a process")**

**print("2) Down a Process")**

**print("3) Send a Message")**

**print("4) Exit")**

**choice = int(input("Enter choice: "))**

**if choice == 1:**

**process\_id = int(input("Enter process id: "))**

**bully.up(process\_id)**

**elif choice == 2:**

**process\_id = int(input("Enter process id: "))**

**bully.down(process\_id)**

**elif choice == 3:**

**process\_id = int(input("Enter process id: "))**

**bully.message(process\_id)**

**else:**

**break**

**ring.py**

**class Ring:**

**def \_\_init\_\_(self, num\_process=5):**

**self.num\_process = num\_process**

**self.coordinator = 5**

**self.active\_processes = set(range(1, num\_process + 1))**

**def election(self, process\_id):**

**if self.coordinator is None:**

***# Only one process in the system***

**self.coordinator = process\_id**

**print(f"Process {process\_id} is the coordinator.")**

**return**

**if process\_id not in self.active\_processes:**

**print(f"Process {process\_id} is not active.")**

**return**

**highest\_id = process\_id**

**next\_process = (process\_id % self.num\_process) + 1**

**while next\_process != process\_id:**

**if next\_process in self.active\_processes:**

**print(**

**f"Process {process\_id} is passing election message to process {next\_process}."**

**)**

**if next\_process > highest\_id:**

**highest\_id = next\_process**

**else:**

**print(**

**f"Process {next\_process} is down and cannot receive the election message."**

**)**

**next\_process = (next\_process % self.num\_process) + 1**

**self.coordinator = highest\_id**

**print(f"Process {self.coordinator} is the coordinator.")**

**def start\_election(self, process\_id):**

**if process\_id not in self.active\_processes:**

**print(f"Process {process\_id} is not active.")**

**return**

**print(f"Process {process\_id} starts the election process.")**

**self.election(process\_id)**

**def bring\_up\_process(self, process\_id):**

**if process\_id in self.active\_processes:**

**print(f"Process {process\_id} is already up.")**

**return**

**self.active\_processes.add(process\_id)**

**print(f"Process {process\_id} is up.")**

**def bring\_down\_process(self, process\_id):**

**if process\_id not in self.active\_processes:**

**print(f"Process {process\_id} is already down.")**

**return**

**self.active\_processes.remove(process\_id)**

**print(f"Process {process\_id} is now down.")**

**if self.coordinator == process\_id:**

**self.start\_election(process\_id)**

**def print\_active\_processes(self):**

**print("Active processes:")**

**for process\_id in self.active\_processes:**

**print(f"Process {process\_id}")**

**def print\_coordinator(self):**

**if self.coordinator is None:**

**print("Coordinator: None")**

**else:**

**print(f"Coordinator: Process {self.coordinator}")**

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

**ring = Ring()**

**while True:**

**print("-------------------------------------")**

**print("1) Start Election")**

**print("2) Bring Up Process")**

**print("3) Bring Down Process")**

**print("4) Print Active Processes")**

**print("5) Print Coordinator")**

**print("6) Exit")**

**choice = int(input("Enter choice: "))**

**if choice == 1:**

**process\_id = int(input("Enter process id to start the election: "))**

**ring.start\_election(process\_id)**

**elif choice == 2:**

**process\_id = int(input("Enter process id to bring up: "))**

**ring.bring\_up\_process(process\_id)**

**elif choice == 3:**

**process\_id = int(input("Enter process id to bring down: "))**

**ring.bring\_down\_process(process\_id)**

**elif choice == 4:**

**ring.print\_active\_processes()**

**elif choice == 5:**

**ring.print\_coordinator()**

**else:**

**break**

**To run the code**

* Open terminal in VS code and execute the following command **python bully.py** for bully algorithm.
* Open another terminal and execute **python ring.py** for ring algorithm.

# Assignment - 7 (Web - Service)

**app.py**

**from flask import Flask, render\_template, request**

**import requests**

**import json**

**app = Flask(\_\_name\_\_)**

**@app.route("/")**

**def home():**

**return render\_template("index.html")**

**@app.route("/calculate", methods=["POST"])**

**def calculate():**

**num1 = int(request.form["num1"])**

**num2 = int(request.form["num2"])**

**operation = request.form["operation"]**

**payload = {"num1": num1, "num2": num2}**

**if operation == "add":**

**url = "http://localhost:5000/add"**

**elif operation == "subtract":**

**url = "http://localhost:5000/subtract"**

**elif operation == "multiply":**

**url = "http://localhost:5000/multiply"**

**elif operation == "divide":**

**url = "http://localhost:5000/divide"**

**response = requests.post(url, json=payload)**

**result = json.loads(response.text)**

**return render\_template("result.html", result=result)**

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

**app.run(debug=True, port=3000)**

**api.py**

**from flask import Flask, request**

**app = Flask(\_\_name\_\_)**

**@app.route("/add", methods=["POST"])**

**def add():**

**data = request.get\_json()**

**num1 = data["num1"]**

**num2 = data["num2"]**

**result = num1 + num2**

**return str(result)**

**@app.route("/subtract", methods=["POST"])**

**def subtract():**

**data = request.get\_json()**

**num1 = data["num1"]**

**num2 = data["num2"]**

**result = num1 - num2**

**return str(result)**

**@app.route("/multiply", methods=["POST"])**

**def multiply():**

**data = request.get\_json()**

**num1 = data["num1"]**

**num2 = data["num2"]**

**result = num1 \* num2**

**return str(result)**

**@app.route("/divide", methods=["POST"])**

**def divide():**

**data = request.get\_json()**

**num1 = data["num1"]**

**num2 = data["num2"]**

**result = num1 / num2**

**return str(result)**

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

**app.run(debug=True)**

Create a folder named **templates** in the root directory of the assignment and add the following two files in it.

**index.html**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>Calculator Web App</title>**

**</head>**

**<body>**

**<h1>Calculator Web App</h1>**

**<form action="/calculate" method="POST">**

**<label for="num1">Number 1:</label>**

**<input type="number" id="num1" name="num1" required /><br />**

**<label for="num2">Number 2:</label>**

**<input type="number" id="num2" name="num2" required /><br />**

**<label for="operation">Operation:</label>**

**<select id="operation" name="operation" required>**

**<option value="add">Addition</option>**

**<option value="subtract">Subtraction</option>**

**<option value="multiply">Multiplication</option>**

**<option value="divide">Division</option></select**

**><br />**

**<input type="submit" value="Calculate" />**

**</form>**

**</body>**

**</html>**

**result.html**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>Calculator Web App - Result</title>**

**</head>**

**<body>**

**<h1>Calculator Web App - Result</h1>**

**<p>The result is: {{ result }}</p>**

**<a href="/">Go Back</a>**

**</body>**

**</html>**

To run the web service follow these steps:

* Open a terminal in VS code and run **python api.py**
* Open another terminal and run **python app.py**
* Open browser and go to **localhost:3000**