BCA\BSc (it) Python Programming Journal Part-4 - Complete Solutions (Exercises 51-63)

Table of Contents

---

51. Two-way TCP Communication

Objective: Create a TCP server and client that can communicate bidirectionally.

Server Program:

```
import
socket

def start_server():
    server = socket.socket(socket.AF_INET,
    socket.SOCK_STREAM) server.bind(('localhost',
    9999)) server.listen()
    print("Server listening on port 9999...")
    client, addr = server.accept()
    print(f"Connected by {addr}")

    while True:
        # Receive message from client msg
        = client.recv(1024).decode('utf-
        8') if not msg or msg.lower() ==
        'exit':
            break
        print(f"Client: {msg}")

        # Send response to client
        response = input("You: ")
        client.send(response.encode('utf
        -8'))

    client.close()
    server.close()

if      __name__      ==
"__main__":
```

`start_server()` **Client**
Program: `import socket`

```
def start_client(): client =
    socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('localhost', 9999))

    while True:
        # Send message to server
        message = input("You: ")
        client.send(message.encode('utf
        -8'))

        if message.lower() == 'exit':
            break

        # Receive response from server
    response = client.recv(1024).decode('utf-
    8') print(f"Server: {response}")
    client.close()
```

```
if __name__ == "__main__":
    start_client()
```

How to Run: 1. Start server in one terminal 2. Start client in another terminal

3. Exchange messages between them **Server**

Output:

```
Server listening on port 9999...
Connected by ('127.0.0.1', 51542)
Client: Hello from
client You: Hi from
server Client: How are
you?
You: I'm good, thanks!
```

Client Output:

```
You: Hello from client
Server: Hi from server
You: How are you?
Server: I'm good, thanks!
You: exit
```

---

52. UDP Server & Client

Objective: Create UDP server and client for connectionless communication.

Server Program: `import`

```
socket

def udp_server(): server =
    socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server.bind(('localhost', 9999)) print("UDP
    Server listening on port 9999...")

    while True:
        data, addr = server.recvfrom(1024)
        message = data.decode('utf-8')
        print(f"Received from {addr}:
        {message}")

        if message.lower() == 'exit':
            break
        response = f"ECHO: {message}"
        server.sendto(response.encode('utf-8'), addr)

if      __name__      ==
"__main__":
udp_server()
```

Client Program:

```
import socket

def udp_client(): client =
    socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_addr = ('localhost', 9999)

    while True: message = input("Enter message: ")
        client.sendto(message.encode('utf-8'),
        server_addr)

        if message.lower() == 'exit':
            break

        response, _ = client.recvfrom(1024)
        print(f"Server response:
        {response.decode('utf-8')}")

if      __name__      ==
"__main__":
udp_client()
```

Server Output:

```
UDP Server listening on port 9999...
Received from ('127.0.0.1', 51542): Hello UDP
Received from ('127.0.0.1', 51542): Test
message   Received   from   ('127.0.0.1',
51542): exit
```

Client Output:

```
Enter message: Hello UDP
Server response: ECHO: Hello UDP
Enter message: Test message
Server response: ECHO: Test message
Enter message: exit
```

---

53. File Server & Client

Objective: Implement file transfer between server and client.

Server Program:

```
import socket
import os

def file_server():
    server = socket.socket(socket.AF_INET,
    socket.SOCK_STREAM) server.bind(('localhost',
    9999)) server.listen()
    print("File Server ready...")

    client, addr = server.accept()
    print(f"Connected by {addr}")

    filename =
    client.recv(1024).decode('utf-8') if
    os.path.exists(filename):
        with open(filename, 'rb') as
            file: while True: data =
            file.read(1024) if not
            data:
                    break
                client.send(data)
        print(f"File {filename} sent successfully")
    else: client.send(b"File not
        found")

    client.close()
    server.close()

if      __name__      ==
"__main__":
file_server()
```

Client Program: `import socket`

```
def file_client(): client =
    socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('localhost', 9999))

    filename = input("Enter filename to request: ")
    client.send(filename.encode('utf-8'))
    data = client.recv(1024)
    if data == b"File not
    found":
        print("File not found on server")
    else:
        with open(f"received_{filename}", 'wb') as file:
            file.write(data) while
            True: data =
            client.recv(1024) if not
            data: break
                file.write(data)
        print(f"File saved as received_{filename}")

    client.close()

if      __name__      ==
"__main__":
file_client()
```

Server

Output:

```
File Server ready...
Connected by ('127.0.0.1', 51542)
File sample.txt sent successfully
```

Client Output:

```
Enter filename to request: sample.txt
File saved as received_sample.txt
```

---

54. Sending Email

Objective: Send email using Python’s smtplib.

Program:

```
import smtplib from email.mime.text
import MIMEText from
email.mime.multipart import
MIMEMultipart

def send_email():
    # Configure these with your actual credentials
    sender = "your_email@gmail.com"
    password = "your_app_password" # Use app-specific password
    receiver = "receiver@example.com"
    # Create message message =
    MIMEMultipart() message["From"] =
    sender message["To"] = receiver
    message["Subject"] = "Test Email from
    Python"

    body = "This is a test email sent using Python's smtplib!"
    message.attach(MIMEText(body, "plain"))

    try:
        # Connect to SMTP server with
        smtplib.SMTP("smtp.gmail.com", 587) as
        server:
            server.starttls()
            server.login(sender, password)
    server.sendmail(sender, receiver, message.as_string())
                        print("Email sent successfully!")
    except Exception as e:
        print(f"Error: {e}")

if __name__ ==
"__main__":
send_email()
```
Output:

```
Email sent successfully!
```

Note: Replace credentials with actual email and app password.

---

55. GUI Application

Objective: Create GUI with buttons, labels, and entry fields.

Program:

```
import tkinter as tk from
tkinter import messagebox

def show_info(): name =
    entry_name.get() age
    = entry_age.get()
    messagebox.showinfo("User Info", f"Name: {name}\nAge:
                                               {age}")
```

```
# Create main window
root = tk.Tk()
root.title("Simple
GUI")
root.geometry("300x2
00")

# Create widgets
label_name = tk.Label(root, text="Name:")
label_name.pack(pady=5)

entry_name = tk.Entry(root, width=30)
entry_name.pack(pady=5)

label_age = tk.Label(root, text="Age:")
label_age.pack(pady=5)

entry_age = tk.Entry(root, width=30)
entry_age.pack(pady=5)

submit_btn = tk.Button(root, text="Submit", command=show_info)
submit_btn.pack(pady=15)
root.mainloop()
```

Output:
GUI Application

---

56. Confirmation Dialogs

Objective: Create confirmation dialogs and alerts.

Program:

```
import tkinter as tk from
tkinter import messagebox

def show_confirmation(): response =
    messagebox.askyesno("Confirmation", "Do you want to proceed?")
    if response: messagebox.showinfo("Result", "You chose Yes!")
    else: messagebox.showinfo("Result", "You
        chose No!")

def show_alert():
    messagebox.showwarning("Important Alert", "This action cannot be
                                          undone!")

root = tk.Tk()
root.title("Dialog
Examples")
root.geometry("300x150")

btn_confirm = tk.Button(root, text="Show Confirmation",
command=show_confirmation) btn_confirm.pack(pady=10)

btn_alert = tk.Button(root, text="Show Alert",

command=show_alert) btn_alert.pack(pady=10)

root.mainloop()
```

Output:
Confirmation Dialog

---

57. Simple Calculator

Objective: Create a simple calculator GUI.

Program:

```
import tkinter as tk

def calculate():
    try: num1 =
        float(entry1.get())
        num2 =
        float(entry2.get())
        operation = var.get()

        if operation == "+":
            result = num1 + num2
        elif operation == "-":
            result = num1 - num2
        elif operation == "*":
            result = num1 * num2
        elif operation == "/":
            if num2 == 0:
                result = "Error: Division by zero"
            else: result = num1 /
        num2 else: result =
        "Invalid operation"

        label_result.config(text=f"Result: {result}")
    except ValueError:
        label_result.config(text="Invalid
        input")

root = tk.Tk()
root.title("Simple
Calculator")
root.geometry("400x250")

# Number inputs
frame_input =
tk.Frame(root)
frame_input.pack(pady=10
)

tk.Label(frame_input, text="Number 1:").grid(row=0,
column=0, padx=5) entry1 = tk.Entry(frame_input)
entry1.grid(row=0, column=1, padx=5)

tk.Label(frame_input, text="Number 2:").grid(row=1,
column=0, padx=5) entry2 = tk.Entry(frame_input)
entry2.grid(row=1, column=1, padx=5)

# Operation selection
frame_ops = tk.Frame(root)
frame_ops.pack(pady=10)
var =
tk.StringVar(value="+")
operations = [("+", "+"), ("-", "-"), ("×", "*"), ("÷", "/")]

for i, (text, op) in enumerate(operations): rb =
    tk.Radiobutton(frame_ops, text=text, variable=var,
    value=op) rb.grid(row=0, column=i, padx=5)

# Calculate button
btn_calculate = tk.Button(root, text="Calculate", command=calculate)
btn_calculate.pack(pady=10)

# Result display
```

```
label_result = tk.Label(root,
text="Result: ")
label_result.pack(pady=10)
root.mainloop()
```

Output:
Calculator GUI

---

58. Database Operations

Objective: List databases and insert rows into a table.

Program:

```
import
sqlite3

def database_operations(): #
    Create connection conn =
    sqlite3.connect('example.db')
    cursor = conn.cursor()

    # Create table
    cursor.execute('''CREATE TABLE IF NOT EXISTS users (
                  id INTEGER PRIMARY
                  KEY, name TEXT NOT
                  NULL, email TEXT NOT
                  NULL)''')

    # Insert rows
    users = [
        ('John Doe', 'john@example.com'),
        ('Jane Smith', 'jane@example.com'),
        ('Bob Johnson',
    'bob@example.com') ]
    cursor.executemany("INSERT INTO users (name, email) VALUES (?, ?)", users)
    conn.commit()

    # Retrieve and display data
    print("Users in database:")
    cursor.execute("SELECT * FROM
    users") for row in
    cursor.fetchall(): print(row)

    # List databases (SQLite specific)
    print("\nDatabase list:")
    cursor.execute("PRAGMA
    database_list") for row in
    cursor.fetchall(): print(f"Database
    {row[0]}: {row[2]}") conn.close()

if __name__ == "__main__":
    database_operations()
```

Output:

```
Users in database:
(1, 'John Doe', 'john@example.com')
(2, 'Jane Smith', 'jane@example.com')
(3, 'Bob Johnson', 'bob@example.com')

Database list:
Database 0: example.db
```

---

59. Update Row in Table

Objective: Update existing rows in a database table.

Program:

```
import

sqlite3

def update_row():
    conn = sqlite3.connect('example.db')
    cursor = conn.cursor()

    # Display before update
    print("Before update:")
    cursor.execute("SELECT * FROM
    users") for row in
    cursor.fetchall(): print(row)

    # Update row
    cursor.execute("UPDATE users SET email = 'john.doe@example.com' WHERE name =
    'John Doe'" conn.commit()
    # Display after update
    print("\nAfter update:")
    cursor.execute("SELECT * FROM
    users") for row in
    cursor.fetchall():
        print(row)

    conn.close()

if __name__ == "__main__":
    update_row()
```

Output:

```
Before update:
(1, 'John Doe', 'john@example.com')
(2, 'Jane Smith', 'jane@example.com')
(3, 'Bob Johnson', 'bob@example.com')

After update:
(1, 'John Doe', 'john.doe@example.com')
(2, 'Jane Smith', 'jane@example.com')
(3, 'Bob Johnson', 'bob@example.com')
```

---

60. Delete Row from Table

Objective: Delete rows from a database table.

Program:

```
import

sqlite3

def delete_row():
    conn = sqlite3.connect('example.db')
    cursor = conn.cursor()

    # Display before deletion
    print("Before deletion:")
    cursor.execute("SELECT * FROM
    users") for row in
    cursor.fetchall():
        print(row)
    # Delete row
    cursor.execute("DELETE FROM users WHERE name = 'Bob Johnson'")
    conn.commit()

    # Display after deletion
    print("\nAfter deletion:")
    cursor.execute("SELECT * FROM
    users") for row in
    cursor.fetchall(): print(row)
    conn.close()

if __name__ ==
"__main__":
delete_row()
```

Output:

```
Before deletion:
(1, 'John Doe', 'john.doe@example.com')
(2, 'Jane Smith', 'jane@example.com')
(3, 'Bob Johnson', 'bob@example.com')

After deletion:
(1, 'John Doe', 'john.doe@example.com')
(2, 'Jane Smith', 'jane@example.com')
```

---

61. Student Database System

Objective: Create student database and table structure.

Program:

```
import
sqlite3

def create_student_db(): conn =
    sqlite3.connect('dbStudent.db')
    cursor = conn.cursor()

    # Create student table
    cursor.execute('''CREATE TABLE IF NOT EXISTS
                   tblStudInfo ( student_id INTEGER
                   PRIMARY KEY AUTOINCREMENT, student_name
                   TEXT NOT NULL, stream TEXT NOT NULL,
                   college_name TEXT NOT NULL,
                   contact_number TEXT,
                    remarks TEXT)''')

    # Insert sample data
    students = [
        ('Rahul Sharma', 'Computer Science', 'ABC College', '9876543210', 'Merit
                                                                    holder'),
        ('Priya Patel', 'Commerce', 'XYZ College', '8765432109', 'Sports
        captain'), ('Amit Kumar', 'Arts', 'PQR College', '7654321098',
        'Cultural head')
    ]

    cursor.executemany('''INSERT INTO tblStudInfo
                    (student_name, stream, college_name, contact_number,
                    remarks) VALUES (?, ?, ?, ?, ?)''', students)

    conn.commit()

    # Display students
    print("Student Database Created
    Successfully!") print("\nStudent
    Records:") cursor.execute("SELECT * FROM
    tblStudInfo") for row in
    cursor.fetchall():
        print(row)

    conn.close()

if __name__ == "__main__":
    create_student_db()
```

Output:

```
Student Database Created Successfully!

Student Records:
(1, 'Rahul Sharma', 'Computer Science', 'ABC College', '9876543210', 'Merit
holder')
(2, 'Priya Patel', 'Commerce', 'XYZ College', '8765432109', 'Sports captain')
(3, 'Amit Kumar', 'Arts', 'PQR College', '7654321098', 'Cultural head')
```

---

62. Update Student Information

Objective: Update student records in the database.

Program:

```
import

sqlite3

def update_student():
    conn = sqlite3.connect('dbStudent.db')
    cursor = conn.cursor()

    # Display before update print("Before
    Update:") cursor.execute("SELECT *
    FROM tblStudInfo") for row in
    cursor.fetchall():
        print(row)

    # Update student student_id = 2
    new_stream = "Business Studies"
    new_contact = "9998887776"
    new_remarks = "Class
    representative"

    cursor.execute('''UPDATE tblStudInfo
                  SET stream = ?, contact_number = ?, remarks = ?
                  WHERE student_id = ?''',
                  (new_stream, new_contact, new_remarks, student_id))

    conn.commit()

    # Display after update print("\nAfter
    Update:") cursor.execute("SELECT *
    FROM tblStudInfo") for row in
    cursor.fetchall():
        print(row)

    conn.close()

if __name__ == "__main__":
    update_student()
```

Output:

```
Before Update:
(1, 'Rahul Sharma', 'Computer Science', 'ABC College', '9876543210', 'Merit
holder')
```

```
(2, 'Priya Patel', 'Commerce', 'XYZ College', '8765432109', 'Sports
captain') (3, 'Amit Kumar', 'Arts', 'PQR College', '7654321098',
'Cultural head')

After Update:
(1, 'Rahul Sharma', 'Computer Science', 'ABC College', '9876543210', 'Merit
holder')
(2, 'Priya Patel', 'Business Studies', 'XYZ College', '9998887776', 'Class
representative')
(3, 'Amit Kumar', 'Arts', 'PQR College', '7654321098', 'Cultural head')
```

---

63. Delete Student Information

Objective: Delete student records from the database.

Program:

```
import

sqlite3

def delete_student(): conn =
    sqlite3.connect('dbStudent.db')
    cursor = conn.cursor()

    # Display before deletion
    print("Before Deletion:")
    cursor.execute("SELECT * FROM
    tblStudInfo") for row in
    cursor.fetchall():
        print(row)

    # Delete student
    student_id = 3
    cursor.execute("DELETE FROM tblStudInfo WHERE student_id = ?",

    (student_id,)) conn.commit()

    # Display after deletion
    print("\nAfter Deletion:")
    cursor.execute("SELECT * FROM
    tblStudInfo") for row in
    cursor.fetchall():
        print(row)

    conn.close()
```

```python
if __name__ == "__main__":
    delete_student()
```

Output:

```
Before Deletion:
(1, 'Rahul Sharma', 'Computer Science', 'ABC College', '9876543210', 'Merit
holder')
(2, 'Priya Patel', 'Business Studies', 'XYZ College', '9998887776', 'Class
representative') (3, 'Amit Kumar', 'Arts', 'PQR College', '7654321098',
'Cultural head')

After Deletion:
(1, 'Rahul Sharma', 'Computer Science', 'ABC College', '9876543210', 'Merit
holder')
(2, 'Priya Patel', 'Business Studies', 'XYZ College', '9998887776', 'Class
representative')
```