SNMP:

AIM:

To write a program to implement a Simple Network Management Protocol(SNMP) server and client using Twisted framework to enable communication between network devices for managing and monitoring purposes.

CODE:

SENDER SIDE:

```python
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol

class SNMPProtocol(DatagramProtocol):
    def datagramReceived(self, data, addr):
        print("Received data from {}: {}".format(addr, data))
        # Process the SNMP request here and prepare the response
        response = "SNMP response"
        self.transport.write(response.encode(), addr)

def run_server():
    reactor.listenUDP(161, SNMPProtocol())
    reactor.run()

# Start the server
run_server()
```

RECEIVER SIDE:

```python
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol

class SNMPClientProtocol(DatagramProtocol):
    def startProtocol(self):
        self.transport.connect('127.0.0.1', 161)
        self.sendRequest()

    def sendRequest(self):
        request = "SNMP request"
        self.transport.write(request.encode())

    def datagramReceived(self, data, addr):
        print("Received data from {}: {}".format(addr, data))
        # Process the SNMP response here

def run_client():
    reactor.listenUDP(0, SNMPClientProtocol())
    reactor.run()

# Start the client
run_client()
```

SAMPLE INPUT/OUTPUT:

SERVER SIDE

```
Received data from ('127.0.0.1', 50287): b'SNMP request'
```

CLIENT SIDE

```
Received data from ('127.0.0.1', 161): b'SNMP response'
```

RESULT: Thus, the Simple Network Management Protocol (SNMP) usingTwisted Python was implemented and tested successfully.

SMTP:

Aim:

To write a program to implement Simple Mail Transfer Protocol (SMTP)using Twisted Python with an aim to develop a reliable email delivery system.

CODE:

```
import smtplib
import ssl

def send_email(sender_email, sender_password, recipient_email, subject, message):
    smtp_server = 'smtp.gmail.com'
    smtp_port = 465

    # Create a secure SSL context
    context = ssl.create_default_context()

    with smtplib.SMTP_SSL(smtp_server, smtp_port, context=context) as server:
        server.login(sender_email, sender_password)

        email_message = f"Subject: {subject}\n\n{message}"
        server.sendmail(sender_email, recipient_email, email_message)
        print("Email sent successfully")


s="oceanly204@gmail.com"
sp="lyihkkozuiqofazp"
sender_email=s
sender_password =sp
r=input("enter receiver mail: ")
recipient_email=r
subject = str(input("Enter Subject: "))
message = str(input("Enter message: "))

send_email(sender_email, sender_password, recipient_email, subject, message)
```

SAMPLE INPUT/OUTPUT:

```
enter receiver mail: poojav1142@gmail.com
Enter Subject: pooja
Enter message: meistergen
Email sent successfully
PS C:\Users\Pooja\Desktop\meistergen>
```

RESULT: Thus, the Simple Mail Transfer Protocol (SMTP) using Twisted Pythonwas implemented and tested successfully.

CoAP:

Aim:

The primary aim of CoAP is to provide a simple and efficient communication protocol for resource-constrained devices in the context of the Internet of Things (IoT). It is designed to enable these devices to easily exchange information in a lightweight and energy-efficient manner.

CODE:

Server:

```python
import asyncio
import aiocoap
from aiocoap import resource, Context, Message

async def handle_coap(request):
    payload = b"Hello, CoAP!"
    return Message(payload=payload, code=aiocoap.numbers.codes.CONTENT)

async def main():
    try:
        # Create CoAP server context
        root = resource.Site()
        root.add_resource(['hello'], handle_coap)

        # Use '::1' for IPv6 loopback address
        context = await Context.create_server_context(root, bind=('::1', 5683))

        print("CoAP server is running.")

        # Run the server indefinitely
        await asyncio.Event().wait()

    except Exception as e:
        print(f"Error in CoAP server: {e}")

if __name__ == "__main__":
    asyncio.run(main())
```

Client:

```python
from aiocoap.numbers.codes import GET
import asyncio
from aiocoap import *

async def coap_client():
    uri = "coap://[::1]/hello"  # Assuming your CoAP server is running on localhost
    request = Message(code=GET, uri=uri)

    # Create CoAP client context
    context = await Context.create_client_context()

    try:
        # Send CoAP request
        response = await context.request(request).response
        print(f"CoAP response code: {response.code}")
        print(f"CoAP response payload: {response.payload.decode('utf-8')}")
    finally:
        # Ensure that the client context is closed
        await context.shutdown()

async def main():
    await coap_client()

if __name__ == "__main__":
    asyncio.run(main())
```

Output:

```
CoAP response code: 2.05 Content
CoAP response payload: Hello, CoAP!
```