KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

EDGE ANALYTICS LABORATORY MANUAL

Subject Code : KGR2358214

Regulation : KGR23

Academic Year : 2023-2024

M. Tech CSE/CS I Year II Sem.

COMPUTER SCIENCE AND ENGINEERING

KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

Affiliated o JNTUH, Chilkur,(V), Moinabad(M) R. R Dist, TS-501504

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

VISION AND MISSION OF THE INSTITUTION

VISION:

To become an institution which is internationally recognized for its holistic approach to engineering, innovative teaching and learning culture, research and entrepreneurial ecosystem, and sustainable social impact in the community.

MISSION:

• To offer undergraduate and post-graduate programs which are supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed in their professional careers.
• To provide faculty and students with an ecosystem that fosters innovation, research, entrepreneurship, and international exposure through strategic partnerships with governmentorganizations and collaboration with industries.
• To provide holistic learning environment to students which will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
• To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic development.

\

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION:

To be recognized as a department of excellence by stimulating a learning environment in which students and faculty will thrive and grow to achieve their professional, institutional and societal goals.

MISSION:

• To provide high quality technical education to students that will enable life-long learning and build expertise in advanced technologies in Computer Science and Engineering.
• To promote research and development by providing opportunities to solve complex engineering problems in collaboration with industry and government agencies.
• To encourage professional development of students that will inculcate ethical values and leadership skills through entrepreneurship while working with the community to address societalissues.

PROGRAM EDUCATIONAL OBJECTIVES

PEO 1: Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles.
PEO 2: Graduates have successful careers in computer science and engineering fields or will be able to successfully pursue advanced degrees.
PEO 3: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism, moral and ethical responsibility.
PEO 4: Graduates will develop the ability to understand and analyze engineering issues in a broader perspective with ethical responsibility towards sustainable development.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

PO I: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO II: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO III: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO IV: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO V: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO VI: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO VII: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of and need for sustainable development.
PO VIII: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO IX: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO X: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO XI: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO XII: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM SPECIFIC OUTCOMES

PSO1: The Computer Science and Engineering graduates are able to analyze, design, develop, test and apply management principles, mathematical foundations in the development of intelligent systems with computational solutions, make them to expert in designing the secure application and hardware prototype
PSO2: The graduating student will be analyze the contemporary research issues in different areasof computer science & engineering and explore research gaps, analyze and carry out research in the specialized/emerging areas.
PSO3: Develop their skills to solve problems in the broad area of programming concepts and appraise environmental and social issues with ethics and manage different projects in multi-disciplinary field to conducive in cultivating skills for successful career, entrepreneurship and higher studies.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

EDGE ANALYTICS LAB

Course Objectives:

1. Understand the concept of edge computing
2. Understand the Edge computing Architecture
3. Implement the edge computing in IOT
4. Understand the concept of multi-access edge computing
5. Implement edge computing in MEC

Course Outcomes:

CO1: Identify the benefits of edge computing

CO2: Develop the microservices in iofog

CO3: Develop user defined services in the edge

CO4: Create use cases in IOT with edge computing

CO5: Develop services in MEC

CO6: Implement use cases in MEC

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

List of Experiments:

1.	Set up the Arduino IDE for ESP8266-12 module and program it to blink a LED light.
2.	Installation tools to create and manage ECN's
3.	Deploy micro services and writing your own microservices
4.	Setup the Communication Parameters
5.	Implement any two Communications protocols
6.	Deploy modules to a Windows IoT Edge device
7.	Create an IoT hub.
8.	Register an IoT Edge device to your IoT hub.
9.	Install and start the IoT Edge for Linux on Windows runtime on your device.
10.	Remotely deploy a module to an IoT Edge device and send telemetry.
11.	Python based basic programs using Raspberry Pi.
12.	Deploy a module Manage your Azure IoT Edge device from the cloud to deploy a module thatsends telemetry data to IoT Hub.
13.	Publishing Data using HTTP.
14.	Sensor Interfacing and Logging using MQTT.
15.	File IO Example - # Example code to demonstrate writing and reading data to/from files
16.	write code to turn on one of the LEDs on the board (Breadboard)

1. Set up the Arduino IDE for ESP8266-12 module and program it to blink a LED light.

Sure, here are the steps to set up the Arduino IDE for ESP8266-12 module and program it to blink a LED light:

1. Download and install the Arduino IDE from the official website: https://www.arduino.cc/en/software

Open the Arduino IDE, go to File -> Preferences, and in the “Additional Boards Manager URLs” field, add the following URL:

2. http://arduino.esp8266.com/stable/package_esp8266com_index.json
3. Go to Tools -> Board -> Boards Manager, search for “esp8266” and install the “esp8266 by ESP8266 Community” package.
4. Connect your ESP8266-12 module to your computer using a USB to serial adapter. Make sure to connect the following pins:
   - VCC to 3.3V
   - GND to GND
   - RX to TX
   - TX to RX
   - CH_PD to 3.3V
   - GPIO0 to GND (for programming mode)
5. Go to Tools -> Board and select “NodeMCU 1.0 (ESP-12E Module)”.
6. Go to Tools -> Port and select the port where your ESP8266-12 module is connected.
7. Copy and paste the following code into the Arduino IDE:

C++
```
void setup() { pinMode(LED_BUILTIN, OUTPUT);  // Initialize the LED_BUILTIN pin as an output }
void loop() { digitalWrite(LED_BUILTIN, HIGH);  // Turn the LED on delay(1000); // Wait for 1
second digitalWrite(LED_BUILTIN, LOW);  // Turn the LED off delay(1000);  // Wait for 1 second }
```

8. Click on the “Upload” button to upload the code to your ESP8266-12 module.
9. After the upload is complete, disconnect GPIO0 from GND and reset your ESP8266-12 module.
10. Your LED should now be blinking every second!

Note: If your LED is not connected to the LED_BUILTIN pin, make sure to change the pin number in the code to the pin where your LED is connected.

2. Installation tools to create and manage ECN's

ECN (Engineering Change Notice) is a document used in the engineering and manufacturing industries to record and communicate changes to a product or system. The tools used to create and manage ECNs can vary depending on the company's needs and workflows, but here are some common tools that may be useful:

1. Product Lifecycle Management (PLM) software: This is a software system that manages the entire lifecycle of a product from conception to retirement. PLM software can be used to create, manage, and track ECNs, as well as manage other product-related information such as CAD files, BOMs, and documentation.
2. Document management software: This is a software system that manages documents electronically, making it easier to store, share, and track changes. Document management software can be used to create, manage, and track ECNs, as well as other documents such as engineering drawings, specifications, and test reports.
3. Electronic workflow software: This is a software system that automates and tracks workflows, making it easier to manage processes such as ECN approvals. Electronic workflow software can be used to create and manage ECN workflows, track approvals, and notify stakeholders of pending changes.
4. Collaboration tools: Collaboration tools such as instant messaging, email, and video conferencing can be used to facilitate communication between engineering teams, project managers, and stakeholders, making it easier to discuss and approve changes.
5. Project management software: This is a software system that helps teams plan, execute, and track projects. Project management software can be used to manage ECN projects, track progress, and ensure that changes are implemented on time and within budget.
6. Change management software: This is a software system that helps companies manage change, including changes to products, processes, and systems. Change management software can be used to create and manage ECNs, track change requests, and ensure that changes are properly implemented and tested.

3. Deploy micro services and writing your own microservices

Microservices are an architectural approach to building applications by breaking down a large application into small, independent services that communicate with each other through APIs. Here are the steps to deploy microservices and write your own microservices:

Deploying Microservices:

1. Choose a cloud platform: The first step to deploying microservices is to choose a cloud platform. Popular cloud platforms for deploying microservices include AWS, Microsoft Azure, and Google Cloud.
2. Create a virtual machine: Once you have chosen a cloud platform, create a virtual machine that will host your microservices.
3. Install Docker: Docker is a popular containerization technology that allows you to package your microservices into containers that can be easily deployed and managed. Install Docker on your virtual machine.
4. Containerize your microservices: Once you have installed Docker, containerize your microservices. This involves creating a Dockerfile that specifies the dependencies and environment for your microservices.
5. Deploy your microservices: Deploy your microservices to your virtual machine using Docker.
6. Configure load balancing: To ensure that your microservices can handle high traffic, configure load balancing to distribute the traffic across multiple instances of your microservices.

Writing Your Own Microservices:

1. Define the service: The first step to writing your own microservice is to define the service. This involves identifying the functionality that the service will provide, the APIs that it will expose, and the data that it will consume and produce.
2. Choose a programming language and framework: Once you have defined the service, choose a programming language and framework that is best suited for the service. Popular programming languages for writing microservices include Java, Python, and Node.js.
3. Create the service: Using the chosen programming language and framework, create the service. This involves writing the code that implements the functionality of the service and exposes the APIs.
4. Containerize the service: Containerize the service using Docker, creating a Dockerfile that specifies the dependencies and environment for the service.
5. Test the service: Test the service to ensure that it works as expected and meets the requirements.
6. Deploy the service: Deploy the service to a virtual machine using Docker, following the steps outlined in the Deploying Microservices section above.
7. Monitor and maintain the service: Monitor the service to ensure that it is functioning correctly and maintain it by fixing bugs, adding new features, and updating dependencies as needed.

4**. Setup the Communication Parameters**

Setting up communication parameters depends on the specific communication method and device you are using. Here are some general steps you can follow to set up communication parameters:

1. Determine the communication method: The first step is to determine the communication method you will use, such as serial, USB, Ethernet, Wi-Fi, or Bluetooth.
2. Identify the communication device: Identify the device you will be communicating with, such as a microcontroller, sensor, or peripheral device.
3. Determine the communication protocol: Determine the communication protocol that the device uses, such as SPI, I2C, UART, or Modbus.
4. Set the baud rate: For serial communication, set the baud rate to match the device's baud rate. The baud rate is the rate at which bits are transmitted over the communication channel.
5. Set the data format: For serial communication, set the data format to match the device's data format. This includes the number of data bits, parity, and stop bits.
6. Configure network settings: For Ethernet or Wi-Fi communication, configure the network settings, including the IP address, subnet mask, gateway, and DNS servers.
7. Set the communication mode: For some protocols, such as SPI and I2C, set the communication mode to master or slave.
8. Test the communication: Test the communication by sending and receiving data between the devices.
9. Adjust communication parameters as needed: If the communication is not working as expected, adjust the communication parameters, such as the baud rate, data format, or network settings, until the communication is working correctly.

5. Implement any two communa=ications protocols

I2C (Inter-Integrated Circuit) Protocol:

The I2C protocol is a popular communication protocol for low-speed communication between microcontrollers and peripheral devices. It uses a master-slave architecture, where the master initiates the communication and the slave responds to the master's requests.

To implement the I2C protocol, follow these steps:

- Connect the SDA (data) and SCL (clock) lines between the master and slave devices.
- Configure the I2C module on both the master and slave devices.
- On the master device, send a start condition to initiate the communication.
- Send the address of the slave device to the master.
- Send the data or command to the slave device.
- The slave device responds with an acknowledgement or data, depending on the command.
- The master can send more data or commands to the slave device, or send a stop condition to end the communication.

MQTT (Message Queuing Telemetry Transport) Protocol:

2. The MQTT protocol is a popular communication protocol for IoT devices that enables low-bandwidth communication between devices and cloud servers. It uses a publish-subscribe architecture, where devices publish messages to topics, and other devices or servers can subscribe to those topics to receive the messages.

To implement the MQTT protocol, follow these steps:

- Connect the device to an MQTT broker server.
- Configure the MQTT module on the device.
- Subscribe to a topic on the MQTT broker server.
- Publish messages to the topic.
- Other devices or servers subscribed to the topic will receive the messages.
- Devices or servers can also publish messages to the topic, and the device will receive those messages.

6. Deploy modules to a windows IOT Edge device

To deploy modules to a Windows IoT Edge device, follow these steps:

1. Set up your development environment: Install the Azure IoT Edge runtime and development tools on your development machine. You'll also need to create an IoT hub in Azure and register your IoT Edge device with the hub.
2. Create an Azure IoT Edge solution: In Visual Studio, create a new Azure IoT Edge solution. This will create a solution with a deployment manifest file (deployment.json) and a module project.
3. Add modules to the solution: Add the modules you want to deploy to the solution. You can add modules from the Azure Marketplace, or create your own custom modules.
4. Configure the deployment manifest: Open the deployment.json file and configure the modules you want to deploy, their images, and their settings.
5. Build the solution: Build the solution in Visual Studio to create the module images.
6. Push the module images to a container registry: Push the module images to a container registry, such as Docker Hub or Azure Container Registry.
7. Deploy the solution to the IoT Edge device: Deploy the solution to the IoT Edge device using the Azure IoT Edge portal or the Azure CLI.
8. Monitor the deployment: Monitor the deployment of the modules using the Azure IoT Edge portal or the Azure CLI. You can view logs and metrics for each module.

7. Create an IOT hub

To create an IoT hub in Azure, follow these steps:

1. Sign in to the Azure portal: Go to portal.azure.com and sign in with your Azure account.
2. Create a new IoT hub: Click on "+ Create a resource" button on the left pane and search for "IoT Hub" or find it under "Internet of Things" in the list. Click on "Create" to create a new IoT hub.
3. Configure the IoT hub: Provide a name for the IoT hub, select the subscription, resource group, and region. Choose the pricing and scale tier that meets your requirements. You can choose from a free tier for testing and development, or paid tiers for production environments.
4. Configure advanced settings: You can configure advanced settings such as device-to-cloud partitions, message routing, and custom endpoints.
5. Review and create the IoT hub: Review the settings and click on "Create" to create the IoT hub. The process may take a few minutes to complete.Get the connection string: Once the IoT hub is created, go to the "Overview" tab and copy the "IoT Hub Name" and "Primary Connection String" values. These values will be used to connect devices and applications to the IoT hub.

8. Register an IOT edge device to your IOT hub

To register an IoT Edge device to your IoT hub in Azure, follow these steps:

1. Set up your IoT Edge device: Set up your IoT Edge device with the required software and configuration. You'll need to install the Azure IoT Edge runtime and configure the device to connect to your IoT hub.
2. Retrieve the device connection string: In the Azure portal, go to your IoT hub and navigate to the "IoT devices" tab. Click on the "Add" button to add a new device. Provide a device ID and any other desired settings, then click "Save". Once the device is created, click on it to view its properties. Under "Connection string - primary key", copy the device connection string.
3. Configure the device: On the IoT Edge device, open the configuration file (/etc/iotedge/config.yaml for Linux-based devices, or C:\ProgramData\iotedge\config.yaml for Windows-based devices) and add the device connection string to the "provisioning" section. Save the file.
4. Restart the IoT Edge runtime: Restart the IoT Edge runtime to apply the configuration changes. On Linux-based devices, run the command "sudo systemctl restart iotedge". On Windows-based devices, restart the "IoT Edge" service in the Services app.
5. Verify the device is connected: In the Azure portal, go to your IoT hub and navigate to the "IoT devices" tab. Find the device you just registered and verify that it is listed and connected.

9. Install and Start the IOT edge for linux on windows runtime on your device

To install and start the IoT Edge for Linux on Windows runtime on your device, follow these steps:

1. Install the Windows Subsystem for Linux (WSL): The IoT Edge for Linux on Windows runtime requires the Windows Subsystem for Linux to be installed on your device. Follow the instructions provided by Microsoft to install WSL on your device.
2. Install Docker for Windows: Install Docker for Windows on your device. You can download it from the Docker website.
3. Enable the "Virtual Machine Platform" feature: Open the Control Panel and go to "Programs" -> "Turn Windows features on or off". Scroll down to "Virtual Machine Platform" and enable it. This will enable Hyper-V on your device, which is required for running the IoT Edge for Linux on Windows runtime.

4. Install the IoT Edge for Linux on Windows runtime: Open a PowerShell console with administrator privileges and run the following command to install the IoT Edge for Linux on Windows runtime.

```
Enable-WindowsOptionalFeature -Online -FeatureNameMicrosoft-Windows-Subsystem-Linux
```

10. **Remotely deploy a module to an IOT edge device and send telemetry**

11. iotedge.exe start **Python based basic programs using Raspberry pi**

12. **Deploy a module manage your Azure IOT edge device from the cloud to deploy a module that sends telemetry data to IOT Hub**

To deploy a module to your Azure IoT Edge device from the cloud that sends telemetry data to IoT Hub, follow these steps:

1. Create an IoT Hub and an IoT Edge device: If you haven't already done so, create an IoT Hub in Azure and register an IoT Edge device with the IoT Hub.
2. Create an IoT Edge module: In the Azure portal, go to your IoT Hub and navigate to the "IoT Edge" tab. Click on the "Add" button to add a new module. Provide a name and any other desired settings, then click "Save".
3. Configure the IoT Edge device: On the IoT Edge device, open the configuration file (/etc/iotedge/config.yaml for Linux-based devices, or C:\ProgramData\iotedge\config.yaml for Windows-based devices) and add the module configuration to the "modules" section. Save the file.

```
- name: <module name>
  image: <module image>
  env:
    <environment variable name>: <value>
```

1. Deploy the module: In the Azure portal, go to your IoT Hub and navigate to the "IoT Edge" tab. Click on your IoT Edge device, then click on the "Set Modules" button. Select the module you just created and click "Next". Review the deployment settings and click "Submit".
2. Verify the module is running: On the IoT Edge device, run the command "sudo iotedge list". You should see the module listed as "Running".
3. Send telemetry data: Use a tool or script to send telemetry data to the IoT Hub. You can use the Azure IoT Device Explorer tool, or write your own script using the Azure IoT SDKs.

13. Publishing Data using HTTP

To publish data using HTTP, you can use the HTTP POST method to send data to a web server or a RESTful API. Here are the general steps to follow:

1. Prepare the data: Prepare the data you want to send in a format that the web server or API can understand. For example, if you are sending JSON data, prepare the data in a JSON format.
2. Choose an HTTP library or tool: Choose an HTTP library or tool to send the data. Popular HTTP libraries include Requests for Python, HttpClient for .NET, and the built-in HttpWebRequest class for C#. There are also many HTTP client tools available, such as cURL and Postman.
3. Build the HTTP request: Build an HTTP POST request using the chosen library or tool. The request should include the data you want to send as the request body, along with any necessary headers. The URL of the web server or API endpoint should also be included in the request.
4. Send the HTTP request: Send the HTTP POST request to the web server or API endpoint using the chosen library or tool. The response from the server or API should be received, and any necessary error handling should be performed.
5. Handle the response: Handle the response from the web server or API. If the response includes data, parse the data and use it as needed.

Here's an example of how to publish data using the Requests library in Python:

python
Copy code

```python
import requests

# Prepare the data
data = {'sensor_id': '1234', 'temperature': 25}

# Build the HTTP request
url = 'https://example.com/api/temperature'
headers = {'Content-Type': 'application/json'}
response = requests.post(url, json=data, headers=headers)

# Handle the response
if response.status_code == 200:
    print('Data published successfully.')
else:
    print(f'Error publishing data: {response.status_code}')
```

14. Sensor Interfacing and Logging using MQTT

To interface with a sensor and log data using MQTT (Message Queuing Telemetry Transport), you can follow these general steps:

1. Set up an MQTT broker: First, you will need to set up an MQTT broker to receive and distribute the sensor data. You can use an open-source MQTT broker like Mosquitto or a cloud-based MQTT broker like AWS IoT.
2. Connect the sensor to a microcontroller: Connect the sensor to a microcontroller like an Arduino or a Raspberry Pi. The specific wiring will depend on the sensor and microcontroller you are using. Be sure to refer to the sensor and microcontroller documentation for wiring instructions.
3. Write a program to read sensor data: Write a program on the microcontroller to read the sensor data. The program should format the data in a way that can be sent over MQTT. You can use libraries like PubSubClient for Arduino or paho-mqtt for Python to interface with MQTT.
4. Publish the sensor data to the MQTT broker: Publish the sensor data to the MQTT broker using the MQTT library you chose. The topic should indicate what the data is and where it is coming from. For example, if you are logging temperature data from a sensor attached to a Raspberry Pi, the topic could be `raspberry-pi/temperature`.
5. Subscribe to the sensor data on another device: Subscribe to the sensor data on another device, such as a computer or smartphone, using an MQTT client. The MQTT client should be connected to the same MQTT broker as the microcontroller.
6. Log the sensor data: Log the sensor data to a file or database on the subscribing device. You can use a programming language like Python to subscribe to the MQTT topic and write the data to a file or database.

Here's an example of how to interface with a temperature sensor using an Arduino and log the data using MQTT and Python:

1. Set up the MQTT broker: Set up an MQTT broker like Mosquitto on a computer or cloud service.
2. Connect the sensor to an Arduino: Connect a temperature sensor like the LM35 to an Arduino board. Connect the VCC and GND pins to the appropriate power and ground pins on the Arduino. Connect the output pin of the sensor to an analog input pin on the Arduino.
3. Write a program to read the sensor data: Write a program in the Arduino IDE to read the sensor data and format it in a way that can be sent over MQTT. Use the PubSubClient library to publish the data to the MQTT broker. Here's an example program:

arduino

```
#include <PubSubClient.h>
#include <Ethernet.h>

const char* ssid = "your_SSID";
const char* password = "your_WIFI_password";
const char* mqtt_server = "your_MQTT_broker_server_address";

EthernetClient ethClient;
PubSubClient client(ethClient);

int sensorPin = A0;
float temperature;

void setup() {
  Serial.begin(9600);
  setupEthernet();
  client.setServer(mqtt_server, 1883);
}

void loop() {
  temperature = analogRead(sensorPin) * 0.48828125;
  client.connect("arduino");
  client.publish("temperature", String(temperature).c_str());
  client.disconnect();
  delay(1000);
}

void setupEthernet() {
  Ethernet.begin(mac);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("connected");
}
```

15. File IO example -#example a=code to demonstrate writing and reading data to/from files.

Here's an example Python code to demonstrate writing and reading data to/from files:

python
Copy code

```
# Writing data to a file with open('data.txt', 'w') as file: file.write('Hello, world!\n') file.write('This is a test file.\n') file.write('1,2,3,4,5\n') file.write('6,7,8,9,10\n') # Reading data from a file with open('data.txt', 'r') as file:for line in file: print(line.strip())
```

In this example, we first open a file named 'data.txt' for writing using the `open` function with mode `'w'`. We then write four lines of text to the file using the `write` method, separated by newline characters. Finally, we close the file using the `with` statement, which ensures that the file is closed even if an exception is raised.

Next, we open the same file for reading using the `open` function with mode `'r'`. We then read the contents of the file using a `for` loop that iterates over the lines of the file. The `strip` method is used to remove the trailing newline character from each line. Finally, we print each line to the console.

16. Write code to turn on one of the LEDs on the board(Breadboard)

```
int ledPin = 13; void setup() { // Set the LED pin as an outputpinMode(ledPin, OUTPUT); } void loop() { // Turn on the LEDdigitalWrite(ledPin, HIGH); // Wait for a short period delay(1000); // Turn off the LED digitalWrite(ledPin, LOW); // Wait for a short perioddelay(1000); }
```

This code defines the `ledPin` variable to hold the number of the pin connected to the LED, and sets up the pin as an output in the `setup` function using the `pinMode` function.

In the `loop` function, the code turns on the LED by setting the voltage on the LED pin to `HIGH` using the `digitalWrite` function. It then waits for a short period of time using the `delay` function, turns off the LED by setting the voltage on the LED pin to `LOW`, and waits for another short period of time.You can modify the duration of the delays to adjust the blinking rate of the LED, or use other functions and sensors to control the LED based on different conditions.