



KAMALDNP

kamaldnp.com • IT Training, Varanasi

PDF created by Raj Kamal using AI

Short Notes — Fundamentals of Internet of Things

BTEUP Diploma in Computer Science & Engineering • 5th Semester • Short Notes
Session 2026-27

Short Revision Notes — prepared by KAMALDNP Academy from the official BTEUP 2026-27 syllabus. This is NOT a BTEUP board paper or publication.

These **short revision notes** cover all five units of Fundamentals of Internet of Things from the official BTEUP 2026-27 syllabus, in syllabus order. Each point gives the definition, main components and example a short answer needs, in plain English.

This subject is a **practicum** in the 2026-27 scheme, so these notes target its **theory (lecture) part, lab work and viva** together. Revise one unit, connect its circuits carefully in the lab, then try the likely question at the end of the unit.

Unit 1 — Introduction to Internet of Things (10 Periods)

Topics: Introduction to IoT; defining IoT; things in IoT; characteristics of IoT; physical design of IoT; logical design of IoT; functional blocks of IoT; IoT protocols; IoT communication models; IoT communication APIs; IoT enabling technologies.

Introduction to IoT; defining IoT; things in IoT

The **Internet of Things (IoT)** is a network of physical objects — **things** — fitted with sensors, software and connectivity, so that they can collect data, exchange it over the internet, and act on it, mostly without human help.

Defining IoT in one line: IoT connects everyday objects to the internet so they can sense, communicate and respond intelligently. A smart home light that a phone switches from anywhere, and a soil sensor that reports moisture to a farmer, are both IoT examples.

Things in IoT means any identifiable object that can be connected: home appliances, vehicles, machines, wearable devices, street lights and even farm animals with tags. Each thing needs three abilities:

- A unique identity, so it can be addressed on the network.
- Sensing or actuating ability, to interact with the physical world.
- Communication ability, to send and receive data.

Characteristics of IoT

The main **characteristics of IoT** asked in exams are:

- **Connectivity** — things stay connected to a network and to each other, through wired or wireless links.
- **Unique identity** — every thing has its own identifier or address, so data can be traced to its source.
- **Sensing and intelligence** — sensors collect real-world data and the system takes decisions, not merely displays values.
- **Dynamic and self-adapting nature** — devices join, leave and adjust to changing conditions automatically.
- **Scale** — an IoT system may contain a few devices or lakhs of devices working together.
- **Heterogeneity** — devices of different makes, types and protocols work in one system.
- **Interoperability and security needs** — different systems must understand each other, and the data flowing between them must be protected.

Write any five characteristics with a one-line explanation each for a complete short answer.

Physical design of IoT

The **physical design** of IoT describes the actual hardware things and the protocols they use to communicate — in simple words, what the system is made of and how its parts talk.

It has two main parts:

- **Things in IoT** — the physical devices themselves: sensor nodes, controllers such as Arduino boards, RFID tags, actuators and everyday objects fitted with electronics. A thing senses or acts, processes a little, and communicates.
- **IoT protocols** — the rules used at different layers for communication between devices, gateways, networks and servers. Protocols decide how data is addressed, packed, sent and secured.

So physical design answers the question of construction: which devices are used, how they are identified, and which protocols connect them from the field up to the server or cloud where the data is stored and viewed.

Logical design of IoT; functional blocks of IoT

The **logical design** of IoT describes the abstract working of the system — the software-level pieces that make the hardware useful. It is usually explained through its **functional blocks**:

- **Device** — the thing that senses or acts and connects to the network.
- **Communication** — the block that carries data between device, gateway and server using suitable protocols.
- **Services** — functions such as device monitoring, data publishing and control commands.
- **Management** — looking after devices, users, configuration and health of the whole system.
- **Security** — authentication, authorisation and protection of data at every stage.
- **Application** — the user-facing part, such as a mobile app or dashboard, where data is seen and commands are given.

Memory line: physical design is about hardware and protocols; logical design is about these functional blocks and how they cooperate.

IoT protocols

IoT protocols are the agreed rules by which devices, gateways, networks and applications exchange data. They are grouped by the layer or job they serve:

- **Link / network level** — protocols that connect devices and carry packets, including Wi-Fi, Bluetooth and cellular links at the access side, and IP-based addressing and routing above them.
- **Transport level** — protocols that deliver data reliably or quickly between two ends.
- **Application level** — lightweight protocols designed for small devices with little power and bandwidth, used to publish sensor data, request services and control devices. Publish-subscribe style messaging is popular here because many receivers can get one sensor's data.

Why special protocols? Ordinary web protocols can be heavy for a tiny battery-powered sensor. IoT protocols are kept **lightweight, low-power and tolerant of slow or unreliable links**, so that small devices can run for months on a battery while still reporting regularly.

IoT communication models

A **communication model** describes the pattern by which the parts of an IoT system exchange data. The standard models are:

- **Request-Response model** — a client asks and the server replies, like opening a web page. Simple, but the client must wait for each reply.
- **Publish-Subscribe model** — a publisher sends data to a broker under a topic; all subscribers of that topic receive it. Sensors publish once and many users or apps can listen — very popular in IoT.
- **Push-Pull model** — producers push data into a queue and consumers pull it when ready, so a slow consumer is not overloaded.

- **Exclusive Pair model** — a continuous two-way connection between one client and one server for fast, ongoing exchange.

Choose the model by need: request-response for simple queries, publish-subscribe when one data source must serve many receivers.

IoT communication APIs; IoT enabling technologies

An **API (Application Programming Interface)** is a defined way for one program to request services or data from another. **IoT communication APIs** let applications talk to devices and platforms without knowing their inner details. Two common styles are:

- **REST-based APIs** — use ordinary web requests (such as GET to read a sensor value and POST to send a command); simple and widely used.
- **WebSocket-based APIs** — keep a live two-way connection open, so updates are pushed instantly when a value changes.

IoT enabling technologies are the supporting technologies that make IoT possible: sensors and actuators, RFID and identification, wireless communication networks, cloud computing for storage and processing, big-data handling for the huge volume of readings, and embedded controllers that run the devices. Name the technology and its role for full marks.

Likely question: Define IoT and explain its characteristics. Differentiate between the physical design and logical design of IoT, naming the functional blocks.

Unit 2 — IoT Devices (12 Periods)

Topics: How electronic devices fit with the Internet of Things and why they are important: breadboard and its internal connections; LED and its connections; tri-colour LED; resistor; introduction to end devices, sensors and actuators; differentiating between different sensor types.

How electronic devices fit with IoT; breadboard and its internal connections

Every IoT system begins with **electronic devices**: sensors observe the physical world, a controller decides, and actuators act. Without these devices there is no data to send and no action to take — that is why they are the foundation of IoT, and why circuits are first practised on a **breadboard** before any permanent soldering.

A **breadboard** is a reusable board with a grid of holes for building and testing circuits without soldering; components and jumper wires simply push in and pull out.

Internal connections to remember:

- The two long outer rails on each side are connected **lengthwise** and are used for power: the + rail for supply and the - rail for ground.
- In the central area, holes are connected in **short vertical groups of five**; holes in one group are the same point.
- The central gap separates the two halves, so a chip can sit across it with separate pins on each side.

LED and its connections; tri-colour LED

An **LED (Light Emitting Diode)** is a two-legged device that gives light when current flows through it in the correct direction. The longer leg is the **anode (+)** and the shorter leg is the **cathode (-)**; the flat edge on the rim also marks the cathode side.

Connections: anode goes towards the supply or the controller pin (through a resistor), cathode goes to ground (GND). If connected in reverse, the LED simply does not glow. A resistor is always placed in series to limit current, otherwise the LED can burn out. Blinking an LED is the first Arduino lab experiment — it proves the board, code upload and circuit are all working.

A **tri-colour (RGB) LED** packs red, green and blue LEDs in one package with a common leg plus one leg for each colour. By switching or mixing the three colours in different brightness, many colours can be produced from a single component.

Resistor

A **resistor** is a basic electronic component that **opposes the flow of current** and thereby controls how much current passes through a part of a circuit. Its value is measured in **ohms (Ω)**.

Why it is essential in IoT device circuits:

- **Current limiting** — a resistor in series with an LED keeps the current within the LED's safe limit; without it the LED may glow too brightly and fail.
- **Voltage division** — two resistors in series can give a smaller, required voltage from a larger supply, useful with some sensors.
- **Pull-up / pull-down use** — a resistor holds a digital input pin at a definite high or low level when a switch is open, so the reading does not float randomly.

The coloured bands printed on a resistor body form a **colour code** that states its value. A resistor has no polarity — it can be connected either way round.

Introduction to end devices, sensors and actuators

End devices are the final points of an IoT system that directly meet the physical world — the sensor nodes, smart objects and machines at the edge of the network. They collect data or carry out actions where the event actually happens.

- **Sensor** — a device that detects a physical quantity such as temperature, light, motion, distance or gas, and converts it into an electrical signal the controller can read. A sensor is an **input** to the system. Example: a temperature sensor reports how hot a room is.
- **Actuator** — a device that converts an electrical command into physical action. An actuator is an **output** of the system. Examples: a relay switching a fan on, a motor turning, or a buzzer sounding.

Working chain to quote in answers: **sensor senses → controller decides → actuator acts**. An automatic night lamp shows all three — a light sensor senses darkness, the controller decides, and the lamp (through its switching device) turns on.

Differentiating between different sensor types

Sensors are differentiated (compared) on clear bases — use these headings whenever a question asks you to differentiate between sensor types:

- **By quantity measured** — temperature sensors, light sensors (such as LDR), humidity sensors, gas sensors, motion (PIR) sensors and distance (ultrasonic) sensors each measure one kind of physical quantity.
- **By output** — **analog sensors** give a continuously varying voltage, read on an analog pin; **digital sensors** give only high/low or coded digital values, read on a digital pin.
- **By power need** — **active sensors** need their own supply and may emit energy to measure, while **passive sensors** only respond to energy already present.
- **By contact** — contact sensors must touch the object; non-contact sensors measure from a distance.

Selection rule: first decide what is to be measured, then match the sensor's range, output type and supply to the controller board being used.

Likely question: Draw and explain the internal connections of a breadboard. Differentiate between sensor types on any four bases, and explain the roles of sensors and actuators with examples.

Unit 3 — IoT Networks (12 Periods)

Topics: Introduction to the components of basic IoT networks; types of network connections and how data travels through them; the role of internet protocols; understanding of microcontrollers/Arduino and communication protocols.

Components of basic IoT networks

A **basic IoT network** connects field devices to the place where their data is stored, viewed and acted upon. Its main components are:

- **End devices / sensor nodes** — sense data or perform actions at the location, as studied in Unit 2.
- **Gateway** — a middle device that collects data from many local devices, may convert between protocols, and forwards the data towards the internet. It also passes commands back to the devices.
- **Network / connectivity** — the communication links, wired or wireless, that carry the data between devices, gateway and server.
- **Server / cloud** — stores the data, processes it and runs the application logic.
- **Application / user interface** — the dashboard or mobile app through which the user watches readings and sends control commands.

Data path in one line: **device → gateway → network → server/cloud → application**, and commands travel back along the same path in reverse.

Types of network connections and how data travels through them

IoT devices connect in two broad ways:

- **Wired connections** — cables such as Ethernet. Reliable and fast, with steady power possible, but fitting cables everywhere is costly and devices cannot move.
- **Wireless connections** — radio links such as Wi-Fi for a local area, Bluetooth for very short range, and cellular networks for wide areas. Easy to install and suitable for moving or remote devices, but they use battery power and can face interference.

How data travels: a sensor value is first produced by the device, packed into small data packets with the addresses of sender and receiver, sent over the local link to a gateway or router, then forwarded step by step across networks to the server, where it is stored and shown to the user. Commands for actuators travel the same route in the opposite direction. Short-range links suit homes and labs; long-range, low-power links suit farms and cities where devices are spread out and run on batteries.

The role of internet protocols

Internet protocols are the standard rules that let devices made by different companies understand each other on one network. Their role in an IoT network is:

- **Addressing** — every connected device gets a unique address (its IP address), so data can be sent to exactly the right device and replies can find their way back.
- **Packet delivery** — data is broken into packets, which are routed step by step from source to destination, possibly by different paths.
- **Reliable or fast delivery** — protocols decide whether every packet must be confirmed and re-sent if lost, or whether speed matters more than perfect delivery, as with live readings.

- **Common language for applications** — application-level rules built on internet protocols let apps request data and send commands in a fixed format.

Without these shared rules, each company's devices would speak a private language and a mixed IoT system could not work at all.

Microcontrollers/Arduino and communication protocols

A **microcontroller** is a small computer on a single chip — processor, memory and input-output pins together — made to control a device. It reads sensor pins, takes decisions by running its stored program again and again, and drives output pins connected to LEDs, relays or motors. **Arduino** is the popular open board-and-software platform built around such a microcontroller, studied in detail in Unit 4; in an IoT network it is the typical controller inside an end device.

Communication protocols at this level fall into two groups:

- **On-board / short link protocols** — rules by which the controller talks to nearby sensor modules and displays over a few wires on the same board.
- **Network protocols** — rules by which the device or its gateway exchanges data with servers and apps over Wi-Fi, Bluetooth or cellular links.

Together they complete the chain: the controller understands its sensors locally, and the network protocols carry the results to the user anywhere.

Likely question: Explain the components of a basic IoT network and describe how data travels from a sensor to the user. Write the role of internet protocols in IoT.

Unit 4 — Arduino (12 Periods)

Topics: Features of Arduino device; Arduino device introduction; components of Arduino board; C Arduino programming language; understanding basics of Arduino IDE; functions; control statements; loops; data types; variables.

Arduino device introduction and features of Arduino device

Arduino is an open-source electronics platform — a microcontroller board plus a free programming software (IDE) — used to build devices that sense and control the physical world. A program written on a computer is uploaded to the board, after which the board runs it on its own, repeatedly, without the computer.

Features of the Arduino device:

- **Easy to learn and use** — simple board, simple C-based language, and ready examples in the IDE.

- **Low cost and open source** — board designs and software are freely available, so boards are cheap and widely supported.
- **Digital and analog pins** — connects switches, sensors, LEDs, relays and motors directly.
- **USB connection** — one cable gives power, uploads the program and allows serial messages back to the computer.
- **Repeat working** — the uploaded program stays in memory and runs again whenever power is applied.

Components of Arduino board

The main **components of an Arduino board** (taking the common UNO-type board used in labs) are:

- **Microcontroller chip** — the brain of the board; it stores the uploaded program and executes it pin by pin.
- **Digital input/output pins** — pins that read or give only HIGH or LOW values, used for LEDs, buttons and relays; some are marked for PWM, giving an average-level output.
- **Analog input pins** — pins that read varying voltages from sensors such as LDR or a temperature sensor and convert them into numbers.
- **Power section** — USB port and power jack for supply, a voltage regulator, and 5V, 3.3V and GND pins to power external components.
- **Reset button** — restarts the program from the beginning.
- **Crystal oscillator** — provides the clock timing for the microcontroller.
- **Indicator LEDs and USB interface** — show power and data transfer during uploading.

C Arduino programming language; basics of Arduino IDE

Arduino programs, called **sketches**, are written in a simplified **C/C++ based Arduino programming language**. A sketch has a fixed frame: `setup()` runs once and `loop()` repeats forever. Ready functions such as `pinMode`, `digitalWrite`, `digitalRead`, `analogRead` and `delay` hide the difficult register-level details, so beginners can control hardware with single lines.

Skeleton of every sketch:

```
void setup() { pinMode(13, OUTPUT); }
void loop() { digitalWrite(13, HIGH); delay(1000); digitalWrite(13, LOW); delay(1000); }
```

Basics of the Arduino IDE: it is the free software where sketches are typed. Its main steps are: select the board and port from the Tools menu, write the sketch, click **Verify** to compile and check errors, then click **Upload** to send the program to the board. The **Serial Monitor** window shows values sent back by the board, which is the main way of testing sensor readings.

Functions in Arduino

A **function** is a named block of statements that performs one job and is executed when it is called. Arduino sketches are built from functions — the two compulsory ones are `setup()` and `loop()`, and the language supplies many ready functions for hardware work:

- **`pinMode(pin, mode)`** — sets a pin as INPUT or OUTPUT.
- **`digitalWrite(pin, value)`** — gives HIGH or LOW on a digital pin; **`digitalRead(pin)`** reads its state.
- **`analogRead(pin)`** — reads a varying sensor voltage as a number; **`analogWrite(pin, value)`** gives a PWM output.
- **`delay(ms)`** — pauses the program for the given milliseconds.

Programmers can also write their own functions to avoid repeating lines — for example a `blink()` function used at many places. Rule from the skeleton: pin settings belong in `setup()`, and the repeating working steps belong in `loop()`, calling the functions in the required order.

Control statements and loops in Arduino

Control statements let an Arduino program take decisions instead of running blindly line by line. The main one is **if-else**: `if (value < 200) digitalWrite(ledPin, HIGH); else digitalWrite(ledPin, LOW);` — the lamp lights only when the sensor reports darkness. switch-case can select one action from several fixed choices, and comparisons use `==`, `>`, `<`, `>=`, `<=` with `&&` and `||` joining conditions.

Loops repeat actions:

- **for loop** — repeats a fixed count, for example blinking an LED five times: `for (int i = 0; i < 5; i++) { ... }`
- **while loop** — repeats while its condition stays true, such as waiting while a button remains unpressed.
- Above all, **`loop()` itself** is the master repetition — the whole program body runs again and again for as long as the board has power, which is how a device keeps sensing continuously.

Data types and variables in Arduino

A **variable** is a named memory location in which the program stores a value that may change — a sensor reading, a count, or a pin number. It is declared with its **data type** and name, for example `int sensorValue = 0;`

Data types used in Arduino sketches:

- **int** — whole numbers; the normal choice for counts, pin numbers and analog readings.
- **float** — decimal numbers, used when a measured value such as temperature needs a fractional part.
- **char** — a single character; **boolean** — only true or false, ideal for switch states.
- **byte** — a small whole number (0 to 255), saving memory for small values; **long** — for large whole numbers.

- **void** — no value, written before `setup()` and `loop()` because they return nothing.

Rules mirror C: declare before use, give meaningful names, and choose the smallest type that safely holds the value, because a small board has limited memory.

Likely question: Explain the features and main components of an Arduino board. Write an Arduino sketch using `setup()` and `loop()` to blink an LED, naming the functions used.

Unit 5 — IoT and M2M (10 Periods)

Topics: Introduction; M2M; difference between IoT and M2M; SDN and NFV for IoT — software defined networking, network function virtualisation; IoT and WoT (Web of Things).

Introduction; M2M

This unit connects IoT with three related ideas — **M2M**, the modern network technologies **SDN and NFV**, and the **Web of Things**. Understanding them shows where IoT came from and how large IoT systems are managed.

M2M (Machine-to-Machine) communication means direct communication between devices or machines **without human intervention**. Machines fitted with sensors and communication modules exchange data with a central system or with each other and take fixed actions on it. Classic examples: a vending machine reporting low stock to the supplier's server, an electricity meter sending readings automatically, and a vehicle tracker sending its position at fixed intervals.

Key points of M2M:

- Usually a **closed, specific-purpose** system built for one job.
- Often works on a point-to-point link between the machine and one server.
- Uses dedicated hardware and, traditionally, private or cellular networks.

M2M is treated as the foundation from which the broader idea of IoT grew.

Difference between IoT and M2M

M2M and IoT are related — M2M provides the basic machine connectivity on which IoT builds — but they differ in scope and working. This comparison is a favourite exam question:

- **Scope:** M2M connects machines for one specific application; IoT connects a huge variety of things into one broad, open system.
- **Communication:** M2M is mostly point-to-point between a device and a fixed server; IoT uses the internet, cloud platforms and many-to-many communication.
- **Data use:** M2M mainly collects and forwards data for a fixed purpose; IoT analyses data in the cloud and combines it with other services to take smarter decisions.

- **Human role:** M2M runs machine-to-machine with little human involvement in a closed system; IoT also serves people directly through apps and dashboards.
- **Scale and openness:** M2M systems are smaller and vendor-specific; IoT is designed for very large scale with open standards.

SDN for IoT — software defined networking

SDN (Software Defined Networking) is a network approach in which the **control of the network is separated from the data-forwarding hardware** and placed in a central software controller. In a traditional network every switch or router makes its own decisions using its fixed inbuilt rules; in SDN, the central controller sees the whole network and tells each device how to forward traffic.

Why SDN suits IoT:

- An IoT network has thousands of devices; managing each device separately is impractical, while a central controller can set rules for all of them at once.
- Rules can be changed in software quickly — for example giving priority to emergency sensor data — without visiting or replacing hardware.
- New devices and services can be added by updating the controller's program, making the network flexible and programmable.

Memory line: SDN = network control moved into central software. It makes a large, changing IoT network manageable from one place.

NFV for IoT — network function virtualisation

NFV (Network Function Virtualisation) means running **network functions as software (virtual machines or containers) on general-purpose servers**, instead of doing each function on a separate dedicated hardware box. Functions such as firewalling, routing support, load balancing and gateway services, which earlier needed their own appliances, are created as software instances whenever and wherever they are needed.

Why NFV suits IoT:

- **Flexibility** — a new network service for a growing IoT deployment is started in software, without buying and installing new hardware.
- **Lower cost and space** — one standard server hosts several functions that earlier needed several boxes.
- **Easy scaling** — when device numbers grow, more software copies of a function are started; when load falls, they are stopped.

SDN and NFV are often used together for IoT: SDN controls how traffic flows, NFV provides the network functions as flexible software along those paths.

IoT and WoT (Web of Things)

The **Web of Things (WoT)** is an approach that brings IoT devices onto the **World Wide Web** using the same well-known web standards and styles that websites and web services already use. Instead of every device speaking a special, separate protocol, a WoT device exposes its data and controls as web resources that an app can read or operate through ordinary web-style requests — similar in spirit to opening a web page or calling a web API.

Relation between IoT and WoT:

- IoT is the wider concept — connecting things and exchanging data by any suitable means.
- WoT is a way of **building IoT on web standards**, so that devices become easy to connect with websites, mobile apps and other services.
- Because web tools and skills are already common, WoT makes integration faster and systems from different makers easier to combine.

In short: IoT connects the things; WoT connects them **through the Web**.

Likely question: Differentiate between IoT and M2M on any five points, and explain how SDN and NFV support large IoT networks.

Quick Revision — One Line Each

- IoT connects identified things with sensors and connectivity so they sense, exchange data and act.
- Physical design = devices (things) plus protocols; logical design = functional blocks that organise the working.
- Functional blocks: device, communication, services, management, security and application.
- Publish-Subscribe suits IoT well — one sensor publishes, many subscribers receive through a broker.
- Breadboard rails run lengthwise for power; central holes connect in vertical groups of five.
- LED longer leg is anode (+); always use a series resistor to limit current; RGB LED mixes three colours.
- Sensor is input (senses), actuator is output (acts): sensor → controller → actuator.
- Data path in an IoT network: device → gateway → network → server/cloud → application.
- Internet protocols give every device a unique address and route its packets to the right destination.
- Every Arduino sketch has setup() once and loop() forever; pinMode sets a pin, digitalWrite drives it.
- M2M is closed and single-purpose; IoT is open, cloud-based and many-to-many — IoT grew out of M2M.
- SDN centralises network control in software; NFV runs network functions as software on standard servers.
- WoT brings things onto the Web using web standards, making them easy to link with apps and services.

