SS3 DIGITAL TECHNOLOGIES LESSON NOTES

FIRST TERM

WEEK 1: REVIEW OF SS2 WORK

1.1 INTRODUCTION

Welcome to SS3 Digital Technologies. This is the final and most advanced year of your secondary school studies in this subject. This term, we will be building on the foundations you learned in SS1 and SS2 to explore advanced programming, networking, cloud technologies, and the Internet of Things (IoT).

This first week is dedicated to a comprehensive review to ensure we all have the same foundational knowledge.

1.2 CORE CONCEPTS REVIEW (Q&A FORMAT)

A. BASIC PROGRAMMING

- **What is a variable?**
  - A named container in memory used to store data (e.g., age = 17, name = "Adebayo").
- **What are the basic data types?**
  - **String** (text, e.g., "Hello"), **Integer** (whole numbers, e.g., 25), **Float/Double** (decimal numbers, e.g., 19.5), **Boolean** (True or False).
- **What is a 'conditional statement'?**
  - A statement that allows your code to make decisions. The if, else if (or elif), and else keywords.
  - *Example:* if (age > 18) { print("You can vote"); } else { print("You are not eligible"); }
- **What is a 'loop'?**
  - A structure that repeats a block of code.
  - **for loop:** Repeats a specific number of times (e.g., for every item in a list).

- **while loop:** Repeats *as long as* a certain condition is true.

B. NETWORKING AND INTERNET

- **What is a computer network?**
  - A collection of two or more computers connected together to share resources (like files, printers, internet).
- **Differentiate LAN, WAN, and MAN.**
  - **LAN (Local Area Network):** Small area (school, home, office).
  - **WAN (Wide Area Network):** Large geographical area (the Internet is the largest WAN).
  - **MAN (Metropolitan Area Network):** A city-sized network.
- **What is a network 'topology'?**
  - The physical or logical layout of a network (e.g., Star, Bus, Ring, Mesh).
- **What is an 'IP Address'?**
  - A unique numerical label assigned to each device on a network, used for identification and communication (e.g., 192.168.1.1).
- **What are 'HTML' and 'CSS'?**
  - **HTML (HyperText Markup Language):** The *structure* of a webpage (the "skeleton").
  - **CSS (Cascading Style Sheets):** The *style* and *presentation* of a webpage (the "skin" or "clothes").

C. DATABASE SYSTEMS

- **What is a database?**
  - An organized, structured collection of data.
- **What is a 'DBMS'?**
  - **Database Management System:** The software used to create, manage, and query a database (e.g., MySQL, Microsoft SQL Server, PostgreSQL).
- **What are 'tables', 'records', and 'fields'?**
  - **Table:** A collection of related data (e.g., a "Students" table).

- **Record (or Row):** A single entry in a table (e.g., all the info for *one* student).
- **Field (or Column):** A single piece of data in a record (e.g., the "FirstName" field).

- **What is 'SQL'?**
  - **Structured Query Language:** The standard programming language used to "talk" to a database (e.g., SELECT * FROM Students WHERE Age > 16;).

EVALUATION (WEEK 1)

1. Write a simple if-else statement to check if a variable score is greater than or equal to 50.
2. What is the difference between a for loop and a while loop?
3. What do the acronyms LAN, HTML, and SQL stand for?

ASSIGNMENT (WEEK 1)

1. Briefly describe the function of the following network devices:
   - Router
   - Switch
   - Firewall
2. Write a short paragraph explaining the relationship between HTML, CSS, and a web browser.

WEEK 2: ADVANCED PROGRAMMING I (FUNCTIONS & ARRAYS/LISTS)

2.1 INTRODUCTION

This week, we move from writing linear scripts to writing *structured* code. The two most important tools for this are **Functions** and **Arrays/Lists**. These tools help us write code that is clean, reusable, and efficient.

2.2 FUNCTIONS

- **What is a Function?**
    - A function is a **reusable block of code** that performs a specific task. You "define" it once and can "call" (or "invoke") it many times.
- **Why use Functions?**
    - **DRY Principle (Don't Repeat Yourself):** Instead of writing the same 10 lines of code in five different places, you write it once inside a function and call it five times.
    - **Abstraction:** You can use a function without needing to know *how* it works, just *what* it does (e.g., you use print() without knowing the complex code that sends text to a screen).
    - **Organization:** Breaks a large, complex program into smaller, manageable, and understandable pieces.
- **Key Concepts of a Function:**

    1. **Definition:** Creating the function using a keyword (like def in Python or function in JavaScript).
    2. **Calling:** Using the function's name to execute its code.
    3. **Parameters (Arguments):** Variables in the function definition that act as *inputs*. When you call the function, you pass *arguments* to these parameters.
    4. **Return Value:** A function can *send a value back* to the code that called it using the return keyword.
- **Code Example (Python syntax for clarity):**
- # 1. DEFINITION of a simple function

```
def greet_student(student_name):
  # student_name is a PARAMETER
  print(f"Welcome to SS3, {student_name}!")
  print("Work hard this term.")

# 2. CALLING the function
greet_student("Tunde") # "Tunde" is the ARGUMENT
greet_student("Aisha") # We can reuse it easily

# 3. Function with a RETURN VALUE
def calculate_area(length, width):
  # length and width are PARAMETERS
  area = length * width
  return area # SENDS this value back

# 4. CALLING a function with a return value
room_area = calculate_area(10, 5) # The value 50 is returned
print(f"The area of the room is {room_area} square meters.")
```

2.3 ARRAYS / LISTS

- **What is an Array/List?**
  - An array (or "list" in Python, "array" in JavaScript) is a single variable that can hold **a collection of multiple items** in an ordered sequence.
- **Why use Arrays?**
  - Imagine needing to store the scores of 50 students. You *could* create 50 variables (score1, score2, score3...score50), which is a nightmare.
  - Or, you could create *one* array: scores = [72, 85, 91, ..., 64]
- **Key Concepts of an Array:**

1. **Elements:** The individual items inside the array.

WEEK 3: ADVANCED PROGRAMMING II (OBJECT-ORIENTED PROGRAMMING)

3.1 INTRODUCTION: A NEW WAY OF THINKING

In SS2, you learned "procedural programming" (writing code as a list of steps or functions). **Object-Oriented Programming (OOP)** is a different *paradigm* (a way of thinking and structuring code).

OOP is about modeling your program around real-world **"objects"** (like a Student, a Car, a BankAccount) that have *properties* and *actions*.

3.2 THE CORE CONCEPTS: CLASS & OBJECT

1. **Class:** A **blueprint** or **template** for creating objects.
    - A class defines the *common properties* and *behaviors* that its objects will have.
    - *Analogy:* The architectural *plan* for a house is the Class. It defines that a house will have "walls," "a roof," and "doors."
2. **Object (or Instance):** A **specific instance** of a class. It is the *actual thing* built from the blueprint.
    - *Analogy:* The *actual house* you build from the plan is the Object. Your house (Tunde's house) and your neighbor's house (Aisha's house) are two different *objects* of the same House *class*.

3.3 ATTRIBUTES & METHODS

- **Attributes:** These are the **properties** or **data** that an object holds. They are variables that belong to the class.
    - *Analogy:* For a Car class, attributes would be color, brand, model, current_speed.
- **Methods:** These are the **actions** or **behaviors** that an object can perform. They are functions that belong to the class.
    - *Analogy:* For a Car class, methods would be start_engine(), accelerate(), brake().
- **Code Example (Python):**
- # 1. DEFINING the Class (the blueprint)

```
class Student:
 # The __init__ method is a special "constructor"
 # It runs when a new object is created
 def __init__(self, name, age, student_id):
  # 2. These are ATTRIBUTES
  self.name = name
  self.age = age
  self.student_id = student_id
  self.is_present = False # A default attribute

 # 3. This is a METHOD (an action)
 def mark_present(self):
  self.is_present = True
  print(f"{self.name} is now marked as present.")

 def introduce_self(self):
  print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# 4. CREATING OBJECTS (instances) from the Class
# We are building the actual "houses" from the blueprint
student1 = Student("Bello Tunde", 17, "SS3-001")
student2 = Student("Fatima Ahmed", 16, "SS3-002")

# 5. ACCESSING attributes and CALLING methods
print(f"Student 1's name is: {student1.name}") # Access attribute
print(f"Student 2's ID is: {student2.student_id}")

student1.introduce_self() # Call method
student2.introduce_self() # Call method

print(f"Student 1 present? {student1.is_present}") # False
```

- student1.mark_present() # Call the method to change the attribute
- print(f"Student 1 present? {student1.is_present}") # True

3.4 THE FOUR PILLARS OF OOP

OOP is built on four major principles:

1. **Encapsulation:**
    - *What:* Bundling the data (attributes) and the code (methods) that acts on that data into a single unit (the object). It also means *hiding* the object's internal details and protecting data from outside interference.
    - *Analogy:* When you drive a car, you use the steering wheel and pedals (public methods). You don't need to (and shouldn't) directly mess with the engine's internal wiring (private data). The car *encapsulates* its own complexity.
2. **Inheritance:**
    - *What:* A mechanism where a new class (the "child" or "subclass") automatically *inherits* all the attributes and methods from an existing class (the "parent" or "superclass").
    - *Analogy:* A SportsCar (child) is a *type of* Car (parent). It inherits all the basic Car properties (wheels, engine) but adds its own (like turbo_charge()).
3. **Polymorphism:**
    - *What:* (Greek for "many forms"). The ability for an object, method, or variable to be treated as more than one type. It allows a single action or method name to be used for different types of objects.
    - *Analogy:* You have a speak() method. For a Dog object, speak() will print("Woof!"). For a Cat object, speak() will print("Meow!"). The *same method name* does *different things* depending on the object.
4. **Abstraction:** (Similar to Encapsulation)
    - *What:* Hiding complex implementation details and showing only the essential features to the user.

- *Analogy:* Your TV remote. You see Volume Up, Power On. You don't see the complex circuitry, infrared signals, or code that makes it work. The complexity is *abstracted away.*

EVALUATION (WEEK 3)

1. In OOP, what is the difference between a "Class" and an "Object"?
2. Use the analogy of a "blueprint" and a "house" to explain your answer to Q1.
3. "Attributes" are to properties as "Methods" are to _____.
4. Which pillar of OOP involves a "child" class inheriting from a "parent" class?

ASSIGNMENT (WEEK 3)

1. Think of a real-world object to model as a class: **"BankAccount"**.
2. On paper, list:
   - Three (3) potential **attributes** this class should have (e.g., account_number, balance).
   - Two (2) potential **methods** (actions) this class should have (e.g., deposit(), withdraw()).
3. (Bonus) Try to write the simple BankAccount class in pseudocode or Python.

WEEK 4: SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

4.1 INTRODUCTION

Great software (like an operating system, a mobile app, or a website) doesn't just "happen." It is engineered. The **Software Development Life Cycle (SDLC)** is a structured, formal process that engineering teams use to design, develop, test, and deploy high-quality software.

Using an SDLC helps to:

- Manage complexity and cost.
- Ensure the final product meets the user's needs.
- Reduce bugs and errors.
- Make the process predictable and repeatable.

4.2 THE 6 PHASES OF THE SDLC

Every SDLC model includes these core phases, though they may have slightly different names.

1. **Phase 1: Planning & Requirement Analysis**
    - **Question:** *What* should we build (and *why*)?
    - **Action:** This is the *most important* phase. It involves gathering requirements from stakeholders (the client, users). What problems must it solve? What features must it have?
    - **Output:** A **Software Requirement Specification (SRS)** document.
2. **Phase 2: Design**
    - **Question:** *How* will we build it?
    - **Action:** Architects and senior developers create the blueprint for the system. This includes:
        - **System Architecture:** (e.g., cloud-based, server/client)
        - **Database Design:** (What tables and fields?)
        - **UI/UX Design:** (User Interface/User Experience: what it will look like and feel like)

- **Output:** A **Design Document** (blueprints, wireframes).

3. **Phase 3: Implementation (Coding)**
   - **Question:** *Build* it.
   - **Action:** This is the phase most people think of as "programming." Developers take the design documents and write the actual code in their chosen programming language (Python, Java, C#, etc.).
   - **Output:** The first version of the software.
4. **Phase 4: Testing & Quality Assurance (QA)**
   - **Question:** Does it *work*? Is it *broken*?
   - **Action:** The software is tested to find and fix bugs.
     - **Unit Testing:** Testing individual components (e.g., does the calculate_area() function work?).
     - **Integration Testing:** Do all the components work *together*?
     - **System Testing:** Testing the entire system as a whole.
   - **Output:** A stable, tested version of the software; a list of fixed bugs.
5. **Phase 5: Deployment**
   - **Question:** How do we get it to the *user*?
   - **Action:** The software is released to the public or the client.
     - For a website: Pushing the code to a live web server.
     - For a mobile app: Publishing it on the Google Play Store or Apple App Store.
   - **Output:** The software is now "live."
6. **Phase 6: Maintenance & Updates**
   - **Question:** How do we *keep it working* and *improve it*?
   - **Action:** This is the longest phase. It involves:
     - Fixing new bugs that are discovered.
     - Adding new features (e.g., Version 2.0).
     - Providing user support.

4.3 SDLC MODELS: WATERFALL VS. AGILE

The 6 phases are the "what." The *model* is the "how." It's the strategy for executing the phases.

A. THE WATERFALL MODEL

- **Concept:** A linear, sequential model. You must *finish* one phase completely before you can *start* the next. It's like a waterfall: water flows down, it can't go back up.
    - Phase 1 -> Phase 2 -> Phase 3 -> Phase 4 -> Phase 5 -> Phase 6
- **Analogy:** Building a physical house. You *must* finish the foundation (Design) before you can build the walls (Implementation). You can't change the foundation plan halfway through.
- **Pros:**
    - Very simple to understand and manage.
    - Good for projects where the requirements are *known* and *fixed* from the start.
- **Cons:**
    - **Highly Inflexible.** If the client changes their mind in Phase 4, it's a disaster.
    - **Very Slow.** The user doesn't see *anything* until the very end (Phase 5).

B. THE AGILE MODEL

- **Concept:** An iterative, incremental model. It's the *opposite* of Waterfall.
- **Process:** Instead of building the *whole* project at once, the team builds it in small, fast pieces called **"sprints"** (usually 1-4 weeks).
- **Cycle:** In *each sprint*, the team goes through *all* the phases: Plan -> Design -> Build -> Test -> Release (a small feature) ...then they start the next sprint.
- **Analogy:** Writing a book, chapter by chapter. You write Chapter 1, get feedback, then write Chapter 2. This is much more flexible than writing the *entire* 300-page book before anyone sees it.
- **Pros:**
    - **Extremely Flexible.** Changes are *welcomed* at the start of each sprint.
    - **Fast Feedback.** The client sees a working (though small) part of the product every few weeks.
- **Cons:**
    - Harder to manage.
    - Less predictable on the final deadline and cost.

EVALUATION (WEEK 4)

1. What does SDLC stand for?
2. List the 6 main phases of the SDLC.
3. Which phase is considered the *most important* because it defines what the software will do?
4. Briefly explain the main difference between the **Waterfall** model and the **Agile** model.
5. Which model (Waterfall or Agile) would be better for a project where the requirements are unknown and expected to change often?

ASSIGNMENT (WEEK 4)

1. Imagine your class has been given a project: **"Build a new app for the school tuck shop to allow students to pre-order their lunch."**
2. Briefly list the activities you would do in **Phase 1 (Requirement Analysis)**. (Hint: Who would you talk to? What questions would you ask?)

WEEK 6: NETWORKING II (IP ADDRESSING, SUBNETTING, PROTOCOLS)

6.1 INTRODUCTION

In SS2, we learned *what* a network is. Now, we learn *how* it works. Communication on the internet is like a global postal service. It only works because of two things:

1. **Addresses (IP Addresses):** So every "house" (device) has a unique address.
2. **Rules (Protocols):** So everyone speaks the same "language" to send and receive "mail" (data).

6.2 IP ADDRESSING

- **What:** An **Internet Protocol (IP) Address** is a unique numerical label assigned to every device (computer, phone, server) on a network. Its purpose is **identification** and **location addressing**.
- **Versions:**
    1. **IPv4 (Internet Protocol v4):**
        - The "old" (but still dominant) version.
        - **Format:** 32-bit number, written as four 8-bit numbers (0-255).
        - **Example:** 172.217.14.228 (This is one of Google's addresses)
        - **Problem:** It only allows for ~4.3 billion addresses. We ran out!
    2. **IPv6 (Internet Protocol v6):**
        - The "new" version, designed to solve the address shortage.
        - **Format:** 128-bit number, written in hexadecimal.
        - **Example:** 2001:0db8:85a3:0000:0000:8a2e:0370:7334
- **Public vs. Private IP:**
    - **Public IP:** Your *one* address on the global internet. Your router gets this from your ISP (MTN, Glo, etc.).
    - **Private IP:** The *internal* addresses used *inside* your home or school network (e.g., 192.168.1.5, 10.0.0.2). These are *not* reachable from the internet. Your router acts as the "gatekeeper."

6.3 SUBNETTING

- **What:** Subnetting is the process of taking *one* large network and logically dividing it into *smaller* sub-networks (subnets).
- **Why Subnet?**
    1. **Security:** You can put the "Accounts" department on one subnet and "Student WiFi" on another. They can't see each other's traffic.
    2. **Organization & Performance:** Reduces "broadcast traffic" (network noise).
- **How it Works: The Subnet Mask**
    - A Subnet Mask is a number that "masks" off a part of the IP address to separate it into two parts: the **Network ID** and the **Host ID**.
    - **IP Address:** 192.168.1.10 (Who is this device?)
    - **Subnet Mask:** 255.255.255.0 (What network is it on?)
    - The 255s mean "This part is the Network." The 0 means "This part is the Host."
    - So, for 192.168.1.10 with mask 255.255.255.0:
        - Network ID: 192.168.1
        - Host ID: 10
    - Any device with the 192.168.1 Network ID is on the *same subnet*.

6.4 NETWORK PROTOCOLS

- **What:** Protocols are the **rules and standards** that allow different devices to communicate. They are the "language" of the internet.
- **The TCP/IP Suite:** This is the *collection* of all the main protocols.
- **Key Protocols to Know:**
    - **IP (Internet Protocol):** The "addressing" protocol. It puts the "To" and "From" address on the data packet.
    - **TCP (Transmission Control Protocol):**
        - The **"reliable"** protocol.
        - It breaks a large file into *numbered packets*, sends them, and *checks* if they all arrived correctly. If not, it resends them.

- **Use Case:** Downloading a file, sending an email (you need 100% of the data).

- **UDP (User Datagram Protocol):**
  - The **"fast"** protocol. It just *sends* the packets and doesn't check.
  - It's "unreliable" but much faster.
  - **Use Case:** Video streaming, online gaming, a live call (if you miss one pixel, who cares? The video must *not* buffer).
- **HTTP (HyperText Transfer Protocol):** The protocol for viewing websites.
- **HTTPS (HTTP Secure):** The *secure*, *encrypted* version of HTTP. (Always look for the 's'!)
- **FTP (File Transfer Protocol):** For uploading and downloading large files.
- **SMTP (Simple Mail Transfer Protocol):** For *sending* email.
- **DHCP (Dynamic Host Configuration Protocol):** The protocol that *automatically* assigns an IP address to your phone/laptop when you join a network.

EVALUATION (WEEK 6)

1. What is the main difference between IPv4 and IPv6? Why was IPv6 created?
2. What is the purpose of a **Subnet Mask**?
3. Explain the difference between your **Public IP** and **Private IP**.
4. Why would you use **TCP** for downloading a file but **UDP** for a video call?
5. What protocol *automatically* gives your phone an IP address at a public hotspot?

ASSIGNMENT (WEEK 6)

1. Find your computer's (or phone's) *private* IP address. (On Windows, type ipconfig in Command Prompt. On Mac, check Network Settings. On phone, check Wi-Fi settings).
2. What does the 'S' in **HTTPS** stand for, and why is it important when you are online shopping or banking?

WEEK 8: CLOUD & VIRTUALIZATION

8.1 INTRODUCTION

This week we explore two of the most powerful technologies in modern computing, which are the foundation of the internet you use every day: Virtualization and Cloud Computing.

8.2 VIRTUALIZATION

- **What:** Virtualization is the process of using software to create a **virtual version** of a physical resource. You can virtualize a computer, a server, a storage device, or an operating system.
- **Key Component: The Hypervisor**
    - A hypervisor (e.g., VirtualBox, VMWare) is the *software* that creates and runs **Virtual Machines (VMs)**.
- **What is a Virtual Machine (VM)?**
    - A VM is an *emulated computer system* that runs *on top* of another physical computer. It's like having a "computer inside your computer."
- **Analogy:**
    - Your physical laptop is the "Host" (e.g., a Windows 11 laptop).
    - You install a hypervisor (like VirtualBox).
    - Inside VirtualBox, you create a VM, which is just a set of files on your hard drive.
    - You can then install a *different* OS (e.g., Linux or macOS) inside that VM.
    - You now have Linux running in a window *inside* your Windows machine.
- **Benefits:**
    - **Efficiency:** One powerful physical server can be "sliced" into 20 small VMs, all running different tasks. This saves money, electricity, and space.
    - **Testing & Safety:** A VM is "sandboxed." You can run a virus or test dangerous software inside a VM, and it *cannot* harm your real (host) computer.

8.3 CLOUD COMPUTING

- **What:** Cloud computing is the on-demand delivery of IT resources (servers, storage, databases, software) *over the Internet* ("the cloud") with pay-as-you-go pricing.
- **How it relates to Virtualization:** Cloud computing *depends* on virtualization. When you "rent" a server from Amazon, you are just renting a VM on one of their giant physical servers.
- **Analogy: Physical Server vs. Cloud Server**
    - **Traditional (On-Premise):** You want to start a website. You must buy a physical server, install it, power it, cool it, and secure it. This is like *buying, fueling, and maintaining your own generator.* It's expensive and a lot of work.
    - **Cloud Computing:** You just "rent" server space from a cloud provider (like Amazon Web Services - AWS, Microsoft Azure, Google Cloud). This is like *using NEPA/PHCN.* You just pay for the electricity you use, and they handle all the generators and maintenance.

8.4 SERVICE MODELS OF CLOUD COMPUTING

This is a "stack" – each level builds on the one below it.

1. **IaaS (Infrastructure as a Service)**
    - **What:** Renting the *raw infrastructure* – virtual servers (VMs), storage, networking.
    - **Who uses it:** System Admins, Network Engineers.
    - **Analogy:** Renting an *empty kitchen* (oven, sink, counters). You must bring your *own ingredients* and *do all the cooking.*
    - **Example:** AWS EC2, Google Compute Engine.
2. **PaaS (Platform as a Service)**
    - **What:** Renting the infrastructure *plus* the tools, operating systems, and databases needed to *build and deploy* applications.
    - **Who uses it:** Software Developers.
    - **Analogy:** Renting a *pizza kitchen* (oven, counters, *plus* pre-made dough, cheese, tomato sauce). You just have to *assemble the pizza* (write your app code).
    - **Example:** Heroku, Google App Engine.

3. **SaaS (Software as a Service)**
    - **What:** Renting a *finished, ready-to-use* software application that runs in your browser.
    - **Who uses it:** The end-user (you!).
    - **Analogy:** *Ordering a pizza*. You don't cook, you just eat.
    - **Example: Gmail**, **Google Docs**, **Microsoft 365**, **Netflix**. You don't own the software; you just use it.

EVALUATION (WEEK 8)

1. What is a Virtual Machine (VM)?
2. What is the name of the software that *creates and runs* VMs?
3. Explain the "Generator vs. NEPA" analogy for Cloud Computing.
4. What do IaaS, PaaS, and SaaS stand for?
5. Is Gmail an example of IaaS, PaaS, or SaaS? Explain why.

ASSIGNMENT (WEEK 8)

1. List three (3) *cloud services* that you personally use every day. (Hint: social media, email, streaming, file storage).
2. For each service, explain *what* data it is storing for you "in the cloud."

WEEK 9: INTERNET OF THINGS (IOT)

9.1 INTRODUCTION

The "Internet of Things" (IoT) is the next evolution of the internet.

- **Internet of People:** You use a device (laptop, phone) to *manually* access the internet.
- **Internet of Things:** Everyday *physical objects* (lights, cars, fridges, watches) are embedded with sensors and Wi-Fi to *automatically* connect to the internet, collect data, and talk to each other.

In short: IoT gives a "digital voice" and "senses" to dumb objects.

9.2 HOW IOT WORKS (THE 4 COMPONENTS)

1. **The "Things" (Sensors/Devices):** The physical object itself, plus a sensor to collect data.
    - *Example:* A "Smart" Lightbulb + a motion sensor.
    - *Example:* A "Smart" Watch + a heart-rate sensor.
2. **Connectivity:** The way the data gets to the internet.
    - *Example:* Wi-Fi, Bluetooth, 4G/5G, LoRaWAN.
3. **Data Processing:** The data is sent "to the cloud" (see Week 8) where it is processed by software.
    - *Example:* The heart-rate data is sent to a server, which analyzes it.
4. **User Interface:** The way *you* interact with the data and control the device.
    - *Example:* A *mobile app* on your phone that shows you your heart-rate history.
    - *Example:* Your *voice* ("Alexa, turn on the lights").

9.3 APPLICATIONS OF IOT

A. SMART HOMES

This is the most common application you will see.

- **Smart Speakers:** (Amazon Echo, Google Home) Control your home with your voice.

- **Smart Lighting:** Lights that turn on automatically when you walk in, or that you can control from your phone when you are on vacation.
- **Smart Security:** Security cameras that send an alert to your phone when they detect motion. Smart locks that you can open with your phone.
- **Smart Appliances:** Fridges that tell you when you're out of milk (in development); smart ACs that learn your schedule.

B. SMART CITIES

This is IoT on a large, public scale, used by a government to manage a city.

- **Smart Traffic Management:** Traffic lights that use *live camera data* to adjust their timing, reducing congestion.
- **Smart Parking:** Sensors in parking spaces that tell an app where the *empty spots* are.
- **Smart Waste Management:** Waste bins with sensors that *signal* the waste truck when they are *full*, so trucks don't waste fuel checking empty bins.
- **Smart Grid:** An electricity grid that can intelligently manage power, reducing blackouts.

C. OTHER MAJOR AREAS

- **Wearables:** Smartwatches (Apple Watch) and fitness trackers (Fitbit) that monitor your health.
- **Smart Agriculture:** Farmers using sensors to monitor *soil moisture* and *exactly* how much water/fertilizer a crop needs, reducing waste.
- **Industrial IoT (IIoT):** Factories using sensors to predict when a machine is *about to break down*, so it can be fixed *before* it fails.

9.4 CHALLENGES OF IOT

- **Security:** This is the #1 problem. Every "smart" device is a "hackable" device. A hacker could take over your smart camera or, worse, a city's smart traffic lights.
- **Privacy:** These devices are *always* listening and *always* collecting data. Who owns that data? What are they doing with it?

- **Connectivity:** In Nigeria, the reliability of Wi-Fi and 5G is a major challenge.

EVALUATION (WEEK 9)

1. In your own words, what is the "Internet of Things"?
2. List the 4 basic components of any IoT system.
3. "Smart Waste Management" is an example of what larger IoT application? (Smart Home, Smart City, or Wearables?)
4. What is the *single biggest challenge* facing IoT technology?

ASSIGNMENT (WEEK 9)

1. Design a *new* (or existing) "Smart" device for a Nigerian home.
2. Describe:
    - What is the device?
    - What *sensor(s)* would it use? (e.g., temperature, motion, light, water)
    - What *problem* would it solve for a Nigerian family?
    - (e.g., A smart water tank sensor that sends an alert to your phone when "NEPA water" is available or when your tank is full/empty).

WEEK 10: DATA PROTECTION & BACKUP SYSTEMS

10.1 INTRODUCTION

In the digital age, **data is the new oil**. It is the most valuable asset for any company and for you.

- **Company Data:** Customer lists, financial records, secret code.
- **Personal Data:** Your photos, bank details, school projects, ID numbers.

What happens if you lose it? A business can fail. A person's life can be ruined. This week, we learn the professional strategies to *protect* and *recover* data.

10.2 BACKUP SYSTEMS

- **What is a Backup?** A *copy* of your data stored in a separate, safe location.
- **The 3-2-1 Rule (The "Gold Standard"):** This is the best practice for backups.
    - **3** copies of your data (the original + 2 backups).
    - **2** different types of storage media (e.g., 1 on your computer, 1 on an external hard drive).
    - **1** copy must be **off-site** (e.g., in the cloud, at your office, in another building). Why? If your house burns down, it takes your computer *and* your external hard drive. The off-site copy is your only protection.
- **Backup Types:**
    - **Full Backup:** Copies *everything*, every time. (Slow, takes lots of space).
    - **Incremental Backup:** Copies *only the files that have changed* since the *last* backup. (Fast, less space).
    - **Differential Backup:** Copies *all files that have changed* since the *last FULL* backup.

10.3 REDUNDANCY (HIGH AVAILABILITY)

- **What:** Redundancy is *not* a backup. Redundancy is about *preventing failure* in the first place by duplicating critical *hardware* components. It's for systems that *cannot* be allowed to fail.

- **Backup vs. Redundancy (Analogy):**
  - A **Backup** is the *spare tire* in your car's trunk. If you get a flat, you have to *stop*, do some work (restore), and then you can drive again.
  - **Redundancy** is a *dual-wheel truck*. It has two tires right next to each other. If one goes flat, the other is *already working* and the truck *doesn't even stop.*
- **Example: RAID**
  - **RAID (Redundant Array of Independent Disks):** Using multiple hard drives as one unit.
  - **RAID 1 (Mirroring):** You have two hard drives. All data is written to *both* drives at the same time. If one drive fails, the other one takes over instantly. There is *zero* downtime.

10.4 DISASTER RECOVERY (DR)

- **What:** A **Disaster Recovery (DR) Plan** is the *complete, formal plan* for what to do when a disaster happens (fire, flood, cyberattack, building collapse). A backup is just *one part* of a DR plan.
- **The plan includes:**
  - Who to call.
  - Where the off-site backups are.
  - How to restore the data.
  - How to set up new servers.
- **Key Metrics in a DR Plan:**

1. **RPO (Recovery Point Objective):**
   - **Question:** How much data can we *afford to lose*?
   - **Answer:** This tells you *how often* to back up.
   - *Example:* If a bank sets an RPO of 1 minute, they must back up their transactions *every minute.* If the RPO is 1 day, a full nightly backup is fine.
2. **RTO (Recovery Time Objective):**
   - **Question:** How *fast* do we need to be *back online*?
   - **Answer:** This tells you how much to *spend* on redundancy.

- *Example:* Jumia's website might have an RTO of 5 minutes (very expensive). A school's website might have an RTO of 24 hours (cheaper).

EVALUATION (WEEK 10)

1. What is the **3-2-1 Rule** of backup?
2. Explain why the "1" (one copy off-site) is so important.
3. What is the difference between a **Backup** and **Redundancy**? (Use the spare tire analogy).
4. What does **RAID** stand for?
5. Define **RPO** and **RTO**. Which one measures *time lost* and which one measures *data lost*?

ASSIGNMENT (WEEK 10)

1. Create a *personal* Disaster Recovery plan for your *schoolwork* (e.g., your WAEC project, your notes).
2. How will you implement the 3-2-1 rule?
    - Where is Copy 1 (Original)?
    - Where is Copy 2 (Local Backup)?
    - Where is Copy 3 (Off-site Backup)? (e.g., 1. On my laptop. 2. On a USB flash drive. 3. Emailed to myself / on Google Drive).

WEEK 11-13: REVISION & EXAMINATIONS

- This period is for a comprehensive review of all topics from the term.
- **Key Connections to Review:**
    - How does **OOP** (Week 3) help organize the code for a complex **SDLC** project (Week 4)?
    - How is **Virtualization** (Week 8) the technology that makes **Cloud Computing** (Week 8) possible?
    - How does the **IoT** (Week 9) rely *heavily* on **Cloud** (Week 8) for data processing and **Networking Protocols** (Week 6) for communication?
    - Why is a **Data Backup & DR Plan** (Week 10) so critical for any **IoT** (Week 9) or **Cloud** (Week 8) company?
- Practice past questions and prepare for the final examination.

SS3 DIGITAL TECHNOLOGIES LESSON NOTES

SECOND TERM

WEEK 1: ADVANCED WEB DESIGN (JAVASCRIPT & RESPONSIVE DESIGN)

1.1 INTRODUCTION

In SS1/SS2, you learned the "Big Two" of web design:

1. **HTML (HyperText Markup Language):** The **structure** or "skeleton" of a webpage.
2. **CSS (Cascading Style Sheets):** The **style** or "skin/clothes" of a webpage (colours, fonts, layout).

This week, we learn the "Big Third": **JavaScript (JS).**

- **JavaScript:** The **behaviour** or "brain/muscles" of a webpage. It makes the website *interactive.*

A house analogy: HTML is the foundation and walls. CSS is the paint and furniture. JavaScript is the electricity, the sliding gate, and the doorbell.

1.2 JAVASCRIPT (JS) INTERACTIVITY

- **What is JavaScript?** JavaScript is a high-level, client-side scripting language. "Client-side" means it runs *in the user's web browser* (like Chrome, Firefox), not on the server.
- **What does it do?** It allows you to **manipulate the HTML and CSS** of a webpage *after* it has loaded. This creates a dynamic, responsive experience for the user.
- **Core Functions of JavaScript:**
    1. **Responding to Events:** An "event" is anything the user does (e.g., onclick, onmouseover, onkeydown, onsubmit). JS "listens" for these events and runs a function in response.

2. **Manipulating the DOM:** The **DOM (Document Object Model)** is the browser's internal map of your HTML page. JS can add, remove, or change *any* HTML element on the page without reloading.
3. **Form Validation:** Instantly checking a user's input *before* they submit a form (e.g., "This email address is invalid," "Password is too short").
4. **Making Asynchronous Requests (AJAX/Fetch):** Loading new data from a server *in the background* without refreshing the page (e.g., loading new tweets, fetching a live score).
5. **Creating Sliders, Pop-ups, and Modals:** Almost any interactive component on a modern website is powered by JS.

- **Simple Code Example (HTML File):**
- <!DOCTYPE html>
- <html>
- <head>
- <title>JS Demo</title>
- <style>
- #message { color: red; font-size: 20px; }
- button { padding: 10px; }
- </style>
- </head>
- <body>
-
- <h1>JavaScript Demo Page</h1>
-
- <!-- 1. HTML elements with unique IDs -->
- This text will change.
- <button id="myButton">Click Me!</button>
-
- <!-- 2. Include JavaScript at the bottom -->
- <script>
- // 3. Select the HTML elements using their IDs

```
    const messageElement = document.getElementById("message");
    const buttonElement = document.getElementById("myButton");

    // 4. Define a function to run
    function changeTheText() {
      messageElement.textContent = "See? The text changed!";
      messageElement.style.color = "green";
    }

    // 5. Listen for the 'click' EVENT on the button
    // When clicked, it will run the changeTheText function
    buttonElement.addEventListener("click", changeTheText);
  </script>

</body>
</html>
```

1.3 RESPONSIVE DESIGN (CSS MEDIA QUERIES)

- **What is Responsive Design?** It is the practice of designing a single website that *adapts* and *responds* to the size of the user's screen. The goal is to make the website look good and be easy to use on *any* device (a large desktop monitor, a laptop, a tablet, and a small mobile phone).
- **Why is it Essential?** Most internet traffic (over 60%) is now on mobile phones. A website that is not responsive will be tiny, unreadable, and impossible to use on a phone, and you will lose most of your users.
- **The Key Tool: The "Viewport" Meta Tag** You *must* put this in your <head> tag. It tells the mobile browser to not "lie" about its width and to render the page at its true device width. <meta name="viewport" content="width=device-width, initial-scale=1.0">
- **The Main Technique: CSS Media Queries** A media query is a CSS feature that allows you to apply different styles *only* when certain conditions are met (like the screen's width).

- **Code Example (CSS File):**
- /* 1. Default styles (for mobile phones first - "Mobile-First" design) */
- body {
- font-size: 16px;
- line-height: 1.5;
- }
-
- .container {
- width: 100%; /* On mobile, use the full width */
- padding: 10px;
- }
-
- .sidebar {
- display: none; /* Hide the sidebar on mobile */
- }
-
- /* 2. MEDIA QUERY: For tablets (e.g., screen is AT LEAST 600px wide) */
- @media (min-width: 600px) {
- .container {
- width: 90%;
- margin: 0 auto; /* Center it */
- }
- .sidebar {
- display: block; /* Show the sidebar again */
- }
- }
-
- /* 3. MEDIA QUERY: For desktops (e.g., screen is AT LEAST 1024px wide) */
- @media (min-width: 1024px) {
- .container {
- width: 80%;

```
    max-width: 1200px; /* Set a max width */
  }
  body {
    font-size: 18px; /* Make text slightly bigger */
  }
}
```

EVALUATION (WEEK 1)

1. Explain the "House Analogy" for HTML, CSS, and JavaScript.
2. What does "client-side" mean in the context of JavaScript?
3. What is the **DOM**, and what does JavaScript do to it?
4. Define "Responsive Design."
5. What is the main CSS technique used to create a responsive design?

ASSIGNMENT (WEEK 1)

1. Using the JavaScript example, write the HTML and JS code for a button that, when clicked, hides a paragraph (Hint: element.style.display = "none";).
2. Write a simple CSS media query that changes the body background colour to lightgrey *only* if the screen is wider than 800px.

WEEK 2: ADVANCED DATABASE MANAGEMENT (SQL BASICS)

2.1 INTRODUCTION

In SS1/SS2, we learned *what* a database is (tables, records, fields). Now, we learn how to *talk* to one. **SQL (Structured Query Language)** is the universal programming language for creating, retrieving, and managing data in a **Relational Database Management System (RDBMS)**.

- **RDBMS Examples:** MySQL, PostgreSQL, Microsoft SQL Server, SQLite.

2.2 CATEGORIES OF SQL COMMANDS

1. **DDL (Data Definition Language):** Defines the *structure* of the database.
    - CREATE: Creates a new database or table.
    - ALTER: Modifies an existing table (e.g., add a column).
    - DROP: Deletes a database or table.
2. **DML (Data Manipulation Language):** Manages the *data* within the tables.
    - INSERT: Adds a new row (record).
    - UPDATE: Modifies data in an existing row.
    - DELETE: Deletes a row.
3. **DQL (Data Query Language):** The most common command, used to *ask questions* and retrieve data.
    - SELECT: Fetches data from tables.

2.3 PRACTICAL SQL QUERIES

Let's imagine we have a table named Students:

Table: Students | StudentID | FirstName | LastName | Age | City | |---|---|---|---|---| | 101 | Tunde | Bello | 17 | Lagos | | 102 | Fatima | Ahmed | 16 | Kano | | 103 | Chinedu | Okafor | 18 | Enugu | | 104 | Aisha | Lawal | 17 | Lagos |

A. SELECT: Retrieving Data (DQL)

- **Get ALL columns and ALL rows:**
- SELECT * FROM Students;

  (Result: The entire table)

- **Get only *specific* columns:**
- SELECT FirstName, LastName, City FROM Students;

  (Result: A 3-column table with all 4 students)

- **Get data with a *filter* using WHERE:**
- SELECT * FROM Students WHERE City = 'Lagos';

  (Result: Only the rows for Tunde and Aisha)

- **Using WHERE with a number:**
- SELECT FirstName, LastName FROM Students WHERE Age > 16;

  (Result: Tunde, Chinedu, and Aisha)

- **Using WHERE with AND or OR:**
- SELECT * FROM Students WHERE City = 'Lagos' AND Age < 18;

  (Result: Only Tunde's row)

B. INSERT: Adding New Data (DML)

- **To add a new student:**
- INSERT INTO Students (StudentID, FirstName, LastName, Age, City)
- VALUES (105, 'Emeka', 'Nwosu', 17, 'Onitsha');

  (Result: The table now has 5 rows)

C. UPDATE: Modifying Data (DML)

- **To correct Chinedu's age (CRITICAL: *Always* use WHERE with UPDATE or yo change *every* row!):**
- UPDATE Students
- SET Age = 19
- WHERE StudentID = 103;

(Result: Chinedu's row now shows Age = 19)

D. DELETE: Removing Data (DML)

- **To delete Emeka's record (CRITICAL: *Always* use WHERE with DELETE!):**
- DELETE FROM Students
- WHERE StudentID = 105;

(Result: The table is back to 4 rows)

E. CREATE TABLE: Defining a New Table (DDL)

- **This is how we would have made the table in the first place:**
- CREATE TABLE Students (
- StudentID INT PRIMARY KEY,
- FirstName VARCHAR(100),
- LastName VARCHAR(100),
- Age INT,
- City VARCHAR(50)
-);
    - INT: Integer
    - VARCHAR(100): A variable-length text string (up to 100 characters).
    - PRIMARY KEY: A special constraint that says StudentID must be *unique* for every row. It's the main identifier.

2.4 DATABASE REPORTS

- A database "report" is simply the *output* of a well-structured query, often presented in a user-friendly format (like a table in an application or a printed summary).
- **Advanced Queries for Reports:**
    - **ORDER BY:** Sorts the results.
    - SELECT * FROM Students ORDER BY LastName ASC; -- (ASC = A-Z)
    - **COUNT:** Counts the number of rows.
    - SELECT COUNT(*) FROM Students WHERE City = 'Lagos'; -- (Result: 2)

EVALUATION (WEEK 2)

1. What does SQL stand for?
2. What are the 3 main categories of SQL commands? (DDL, DML, DQL)
3. Write the SQL command to get *only* the FirstName and Age of all students from the Students table.
4. Write the SQL command to find *all* information about the student whose StudentID is 102.
5. What is the critical keyword you *must* use with UPDATE and DELETE to avoid changing the whole table?

ASSIGNMENT (WEEK 2)

1. Using the Students table:
2. Write the *full* SQL query to INSERT a new student: 106, David, Mark, 16, Abuja.
3. Write the *full* SQL query to UPDATE David Mark's Age to 17.
4. Write the *full* SQL query to SELECT all students who are from Lagos *or* Kano.

WEEK 3: NETWORKING III (NETWORK SECURITY & VPNS)

3.1 INTRODUCTION

A network is a "highway" for data. Network security is the practice of protecting that highway—and the "cargo" (data) on it—from thieves, spies, and vandals (hackers). As our entire lives (banking, social, health) move online, network security is no longer optional; it is critical.

3.2 THE "CIA" TRIAD OF SECURITY

The goal of all security is to protect three things (the "CIA" Triad):

1. **Confidentiality:** Keeping data *secret* and private. Only authorized users can *see* it. (Achieved by: Encryption, Passwords)
2. **Integrity:** Keeping data *accurate* and trustworthy. Only authorized users can *change* it. (Achieved by: Hashes, Digital Signatures)
3. **Availability:** Ensuring data and services are *online* and accessible when needed. (Achieved by: Firewalls, Redundancy, DDoS protection)

3.3 KEY SECURITY TOOLS & THREATS

A. FIREWALLS

- **What:** A network security device (hardware or software) that acts as a "gatekeeper" or "bouncer" for a network.
- **Function:** It *monitors* and *filters* all incoming and outgoing network traffic. It follows a set of **rules** to decide what traffic to **Allow** and what traffic to **Block**.
- **Analogy:** A firewall is the security guard at the gate of your school. They check the ID of everyone trying to enter (incoming) and can stop a student from leaving with stolen property (outgoing).

B. COMMON THREATS

- **Malware (Malicious Software):** The "umbrella" term for any software designed to c harm.
    - **Virus:** Attaches itself to a clean file (like a parasite) and spreads when the file is opened.
    - **Worm:** A standalone program that *self-replicates* across a network (like a real worm), consuming bandwidth and infecting other computers.
    - **Ransomware:** *Encrypts* all your files, making them unreadable. The hacker then demands a "ransom" (payment) in Bitcoin to give you the decryption key.
    - **Spyware:** Secretly monitors your activity (keystrokes, webcam) to steal passwords or personal info.
- **Phishing ("Fishing"):**
    - A "social engineering" attack. The hacker *impersonates* a trusted brand (like your bank or Google) and sends you a fake email or text message.
    - The email creates *urgency* ("Your account has been locked! Click here to fix it.")
    - The link leads to a *fake website* that *looks real*. When you enter your username and password, the hacker steals it.
- **Denial of Service (DoS / DDoS) Attack:**
    - An attack on **Availability**.
    - The hacker uses a "botnet" (an army of thousands of infected computers) to flood a target server (like Jumia's website) with *millions* of fake requests.
    - The server is overwhelmed and crashes, becoming "unavailable" to real customers.

3.4 VPNS (VIRTUAL PRIVATE NETWORKS)

- **What is a VPN?** A VPN is a technology that creates a **secure, encrypted connection** (a "tunnel") over a *public, untrusted network* (like the internet).
- **How it Works (Analogy):**
    - **Without a VPN:** You are driving your car on a public road. Anyone watching (your ISP, hackers at a public Wi-Fi hotspot) can see *what car* you are driving (your IP address) and *where* you are going (what website you are visiting).
    - **With a VPN:** A VPN builds a *private, underground, encrypted tunnel* from your computer to the VPN server.

1. Your data is *encrypted* (scrambled) *before* it leaves your computer.
2. It travels through the tunnel to the VPN server (e.g., in Switzerland).
3. The VPN server *decrypts* your data and sends it to the final website (e.g., Google).

- **The "3 C's" - What a VPN Does:**

1. **Concealment (Privacy):** It **hides your real IP address** and replaces it with the IP address of the VPN server. This makes you anonymous and "hides" your location.
2. **Confidentiality (Security):** It **encrypts your traffic**. This is *CRITICAL* on public Wi-Fi (e.g., at an airport or cafe). Without a VPN, a hacker on that Wi-Fi can easily "sniff" (steal) your passwords. With a VPN, all they see is scrambled gibberish.
3. **Circumvention (Access):** It allows you to *bypass* geo-restrictions. A website (like Netflix) sees you are coming from the Swiss server, so it shows you Swiss content, not Nigerian content.

EVALUATION (WEEK 3)

1. What are the three components of the "CIA" Triad of security?
2. What is the main job of a Firewall?
3. Explain the difference between a **Virus** and a **Worm**.
4. What is a **Phishing** attack?
5. Explain the "encrypted tunnel" analogy for a VPN.
6. List the "3 C's" (benefits) of using a VPN.

ASSIGNMENT (WEEK 3)

1. You receive an email from "First Bank" (a bank you don't even use) saying "URGENT: Your account has been suspended due to unusual activity. Click https://www.google.com/search?q=www.firstbank-login-portal.com to verify."
2. List three (3) "red flags" (clues) that this is a phishing attack.

WEEK 4: DIGITAL PROJECT MANAGEMENT

4.1 INTRODUCTION

A "project" is a temporary effort to create a unique product, service, or result (e.g., building a website, developing an app, setting up a school network). **Project Management** is the *discipline* of planning, organizing, and managing all the resources (people, time, money) to successfully complete the project on time, within budget, and to the required quality.

In SS3 (Term 1, Week 4), we studied the **SDLC (Software Development Life Cycle)**, which is the *process* of building software. Project Management is the *management* of that process.

4.2 THE "IRON TRIANGLE" (TRIPLE CONSTRAINT)

Every project is managed by balancing three competing constraints. You cannot change one without affecting the others.

1. **Scope:** *What* are we building? (The features, the tasks, the quality).
2. **Time:** *How long* will it take? (The schedule, the deadline).
3. **Cost:** *How much* will it cost? (The budget, the resources).

- **The Rule:**
    - Want it *faster* (Time ↓)? You must either pay more (Cost ↑) or cut features (Scope ↓).
    - Want *more features* (Scope ↑)? You must add more time (Time ↑) or pay more (Cost ↑).
    - Want it *cheaper* (Cost ↓)? You must take longer (Time ↑) or cut features (Scope ↓).
- **The Project Manager's Job:** To balance this triangle.

4.3 TEAMWORK IN DIGITAL PROJECTS

Digital projects are *never* built by one person. They require a team of specialists.

- **Project Manager (PM):** The leader. Manages the schedule, budget, and team.
- **UI/UX Designer:** Designs *how* the product will look and feel.
- **Frontend Developer:** Builds the part the user *sees* (HTML, CSS, JS).
- **Backend Developer:** Builds the "engine" (server, database, logic).
- **QA (Quality Assurance) Engineer:** Tries to *break* the product (testing) to find bugs.

Key to Teamwork:

- **Communication:** Regular meetings (e.g., "Daily Stand-ups" in Agile).
- **Collaboration:** Using tools that let everyone work on the same project.
- **Version Control:** A system (like **Git**) that tracks *every single change* to the code, so if someone makes a mistake, you can "rewind."

4.4 PROJECT MANAGEMENT TOOLS

A PM doesn't just use a notebook. They use specialized software to track everything.

A. KANBAN BOARDS (e.g., TRELLO)

- **Concept:** A simple, visual way to manage tasks. It's a "pull" system.
- **How it works:** You have a board with *columns* (lists) that represent stages of a workflow.
    - To Do (The list of all tasks)
    - Doing (What the team is working on *right now*)
    - Done (Completed tasks)
- **Tasks** are "cards" that you *move* from left to right.
- **Best for:** Agile teams, small projects, visual thinkers.

B. GANTT CHARTS (e.g., MS PROJECT, ASANA)

- **Concept:** A complex *bar chart* that visualizes the entire project *schedule* over *time*.
- **How it works:**
    - Every task is a horizontal bar.
    - The *length* of the bar shows how long the task will take.
    - It shows **dependencies** (e.g., you can't "Test" until "Build" is finished).

- **Best for:** Waterfall projects, large complex schedules, managing deadlines.

EVALUATION (WEEK 4)

1. Define "Project Management."
2. What are the three points of the "Iron Triangle" (Triple Constraint)?
3. A client tells you, "I need the project finished in half the time, but I can't pay you more." According to the Iron Triangle, what *must* happen?
4. Explain the "To Do / Doing / Done" workflow of a **Kanban Board**.
5. What is the main difference between a **Gantt Chart** and a **Kanban Board**? (Hint: one is about *time*, one is about *flow*).

ASSIGNMENT (WEEK 4)

1. Go to a free Kanban website (like Trello.com) or draw one on paper.
2. Create a board for your "SSCE Exam Preparation."
3. Create three lists: Subjects To Study (To Do), Subjects I Am Studying (Doing), Subjects I Have Mastered (Done).
4. Add "cards" for 5-6 subjects (Maths, English, Digital Tech, etc.) and place them in the correct list.

WEEK 5: MIDTERM TEST

This week is for assessment of all topics covered in Weeks 1-4 (JS/Responsive, SQL, Security/VPN, Project Management).

WEEK 6: EMERGING TECH III (QUANTUM & NANO)

6.1 INTRODUCTION

This week, we look *beyond* today's technology to the "deep future." These are technologies, based on fundamental physics and chemistry, that will change *everything*.

6.2 QUANTUM COMPUTING

- **Classical Computing (Your Laptop):**
  - Works on **"Bits"**.
  - A Bit is a simple switch. It can *only* be a **0** (Off) OR a **1** (On).
  - It's powerful, but it can only do *one thing at a time* (sequentially).
- **Quantum Computing:**
  - Works on **"Qubits" (Quantum Bits)**.
  - Thanks to a weird quantum-physics property called **"Superposition"**, a Qubit can be a **0**, a **1**, or **BOTH at the same time**.
  - This allows a quantum computer to perform *millions of calculations simultaneously*.
- **Analogy: Finding a Book in a Library**
  - **Classical Computer:** It's like you walking down *every single aisle*, one by one, until you find the book. (Takes a long time).
  - **Quantum Computer:** It's like you can instantly check *every aisle in the entire library at the exact same moment*. (Incredibly fast).
- **What will it be used for?**

1. **Breaking Encryption:** This is the *scary* one. A large quantum computer could *instantly* break the encryption (like HTTPS, VPNs) that protects all global banking and military secrets.
2. **Drug & Materials Discovery:** Simulating complex molecules to invent new medicines or materials.
3. **Complex Simulations:** Modeling climate change, or complex financial markets.

- **Current Status:** Highly experimental. They are massive, super-cooled, and very unstable. We are *not* going to have a "Quantum Laptop" anytime soon.

6.3 NANOTECHNOLOGY

- **What is it?** Technology, engineering, and science conducted at the **nanoscale**.
- **What is the "Nanoscale"?** It is the "atomic" or "molecular" level. A "nanometer" (nm) is **one-billionth of a meter**.
    - A human hair is ~80,000 nm wide.
    - A DNA strand is ~2.5 nm wide.
- **Analogy:** It is *manufacturing* and *engineering* with *individual atoms* as your building blocks.
- **What is it used for? (Applications are already here!)**

1. **Medicine (Nanomedicine):**
    - **Drug Delivery:** Designing "nanobots" or "nanoshells" that can carry a drug (like chemotherapy) *directly* to a single cancer cell and ignore the healthy cells.
2. **Materials Science:**
    - **Carbon Nanotubes:** A material 100x stronger than steel but 1/6th the weight. Used in airplanes, stronger sports equipment.
    - **Nanoparticle Coatings:** Creating surfaces that are self-cleaning, anti-bacterial, or super-hydrophobic (water-repellent) for use in paints, glass, and clothing.
3. **Electronics:**
    - Making computer chips *smaller, faster, and more power-efficient* by building them at the nanoscale.

EVALUATION (WEEK 6)

1. What is the difference between a classical "Bit" and a "Qubit"?
2. What is "Superposition"?
3. Explain the "library" analogy for Classical vs. Quantum computing.
4. What is the "nanoscale"? (How many nanometers in a meter?)
5. List two current or future applications of Nanotechnology.

WEEK 8: REVISION OF SSCE PAST QUESTIONS I (THEORY)

8.1 INTRODUCTION

For the next two weeks, our goal is 100% SSCE (WASSCE/NECO) preparation. We will move from *learning* new topics to *applying* all our knowledge from SS1, SS2, and SS3.

This week, we focus on **Theory (Essay-Style) Questions**. These questions test your *depth* of knowledge and your ability to *explain* concepts clearly.

8.2 STRATEGY FOR THEORY QUESTIONS

1. **Read the Verb:** Pay attention to the *action word*.
    - **"List" / "State":** Give a simple list. DO NOT write an essay.
    - **"Define":** Give a clear, one-sentence definition.
    - **"Explain" / "Describe" / "Discuss":** This is where you write. Define the term, then explain it, and *always use an example.*
    - **"Differentiate":** Give a head-to-head comparison, ideally in a table.
2. **Use Keywords:** Your examiner is looking for the *keywords* you learned. (e.g., For "Firewall," they want "monitors," "filters," "incoming/outgoing," "rules").
3. **Be Clear, Not Long:** Be concise. A clear, 3-sentence answer is better than a confused 10-sentence one.

8.3 WORKED EXAMPLES (SS1-SS3 TOPICS)

Q1. (a) What is a computer network? (b) List and explain two (2) types of networks.

- **(a) What is a computer network?**
    - *Answer:* A computer network is a collection of two or more computers and devices, connected together by a communication medium (like cables or Wi-Fi), for the purpose of sharing data and resources (like printers or an internet connection).
- **(b) List and explain two (2) types of networks.**
    - *Answer:*

1. **LAN (Local Area Network):** This network covers a *small* geographical area, such as a single office, a school, or a home. The devices are physically close, and the network is usually privately owned.
2. **WAN (Wide Area Network):** This network covers a *large* geographical area, such as a city, country, or even the whole world. It connects multiple LANs together, often using public infrastructure like telephone lines or satellites. The Internet is the world's largest WAN.

Q2. (a) Define 'Database'. (b) Explain the following terms with an example: (i) Field (ii) Record

- **(a) Define 'Database'.**
  - *Answer:* A database is a structured, organized collection of data, stored and accessed electronically, and managed by a Database Management System (DBMS).
- **(b) Explain (i) Field (ii) Record**
  - *Answer:*
    - **(i) Field:** A Field is a *single column* in a database table. It represents one specific piece of data for all records. *Example:* In a "Students" table, "FirstName" would be a field.
    - **(ii) Record:** A Record is a *single row* in a database table. It represents one complete entry or item, containing all the field data for that item. *Example:* The row "101, Tunde, Bello, 17, Lagos" is one record.

Q3. (a) What is JavaScript? (b) State three (3) functions of JavaScript on a webpage.

- **(a) What is JavaScript?**
  - *Answer:* JavaScript is a client-side scripting language that runs in the user's web browser, used to make webpages dynamic and interactive.
- **(b) State three (3) functions:**
  - *Answer:*
    1. To respond to user events (like clicks or mouse movements).

2. To validate data in forms *before* they are submitted.
3. To manipulate the DOM (change HTML content or CSS styles) without reloading the page. (Other answers: Create pop-ups, sliders, load data in the background).

Q4. Differentiate between a Virus and a Worm.

- *Answer:* | Feature | Virus | Worm | |---|---|---| | **How it Spreads** | Requires a host file (e.g., .exe, .doc) and human action (e.g., opening the file) to spread. | Can self-replicate and spread automatically across a network from one computer to another, exploiting a vulnerability. | | **Standalone** | No, it is a piece of code *attached* to a legitimate program (parasitic). | Yes, it is a standalone program that runs on its own. |

EVALUATION / ASSIGNMENT (WEEK 8)

- **Answer the following SSCE-style questions:**
    1. (a) What is the "Internet of Things" (IoT)? (b) Describe two (2) applications of IoT (one in the home, one in a city).
    2. (a) What does OOP stand for? (b) Differentiate between a "Class" and an "Object."
    3. (a) What is a VPN? (b) State two (2) major benefits of using a VPN on public Wi-Fi.

WEEK 9: REVISION OF SSCE PAST QUESTIONS II (CASE STUDIES / APPLIED ICT)

9.1 INTRODUCTION

This week, we practice the *hardest* type of question: the **Case Study** or **Applied ICT** question. These questions don't just ask you to *define* a term; they give you a real-world *scenario* and ask you to *apply* your knowledge to solve a problem.

9.2 STRATEGY FOR CASE STUDIES

1. **Read and Identify:** Read the scenario carefully. Identify the *client*, the *problem*, and the *goals*.
2. **Select the *Best* Tool:** The question is testing your *judgement*. Don't just list every technology you know. Choose the *most appropriate* one.
    - Problem: "Needs to share files in an office" -> **LAN**
    - Problem: "Needs a dynamic, interactive website" -> **JavaScript**
    - Problem: "Needs to store customer data" -> **Database (RDBMS)**
    - Problem: "Needs to secure the office network" -> **Firewall**
3. **Justify Your Answer:** *Why* did you choose that tool? Explain *how* it solves the client's *specific* problem.

9.3 WORKED EXAMPLES (CASE STUDIES)

Case Study 1: The New Business

- **Scenario:** "Mrs. Okoro is starting a new small business with 15 employees in one building. She wants her staff to be able to share files, access a single customer database, and share one printer. She also needs an internet connection, but is very worried about hackers."
- **Question:** "Advise Mrs. Okoro on the four (4) most important hardware and software components she will need to set up her office network."
- **Answer Strategy:**
    1. **Problem:** Needs to connect 15 staff in *one building* (this screams "LAN"). Needs to *share resources* (files, printer, database). Needs *security*.
    2. **My Answer:**
        - **1. A LAN (Local Area Network) setup:** She will first need a **Switch** and **Ethernet Cables** to physically connect all 15 computers and the printer. This creates the LAN.
        - **2. A Server:** Instead of a simple "peer-to-peer" network, she should have a central **Server**. This server will *host* the customer **Database (using a DBMS like MySQL)** and the shared files, making them easy to back up and manage.

- **3. A Router:** This device is essential. It will manage the LAN and, most importantly, provide the **Internet connection** to all 15 staff by connecting to her ISP.
- **4. A Firewall:** Because she is "worried about hackers," a hardware **Firewall** (often included in the router) is critical. This will act as a gatekeeper, *blocking* unauthorized access from the internet to her private office network, thus protecting her customer database.

Case Study 2: The E-Commerce Website

- **Scenario:** "A fashion designer wants to build a new e-commerce website to sell clothes. She has the following requirements:
  - It must look good on phones and desktops.
  - Users must be able to create an account and log in.
  - The website must show live inventory (how many items are left).
  - She needs to track all her sales and customers."
- **Question:** "List and describe the four (4) key technologies that will be required to build this website."
- **Answer Strategy:**

1. **Problem:** E-commerce site. Needs to be *responsive*, *interactive*, and *data-driven*.
2. **My Answer:**
   - **1. HTML/CSS with Responsive Design:** The foundation. HTML will structure the product pages, and CSS will style them. Critically, **CSS Media Queries** must be used to ensure the site is *responsive* and looks good on phones, as she requested.
   - **2. JavaScript:** This is essential for the *interactive* parts. It will be used for form validation (on the "create account" page) and for creating image sliders to show off the clothes.
   - **3. A Backend Language (e.g., Python/PHP):** She needs a server-side language to handle logic that can't be done in the browser, such as

processing payments, *verifying passwords* at login, and *checking the inventory*.

- **4. A Database (e.g., MySQL with SQL):** This is the heart of the store. A database is required to store *all* the data, including **user accounts** (usernames, hashed passwords), **product details** (price, inventory count), and **sales records**. The backend language will use **SQL** to UPDATE the inventory after a sale.

EVALUATION / ASSIGNMENT (WEEK 9)

- **Answer the following SSCE-style Case Study:**
- **Scenario:** "The Principal of your school wants to reduce paper waste and improve communication. He wants a new system where:
    1. Student results can be recorded and checked online by parents.
    2. The school can send alerts to parents' phones.
    3. Students can submit assignments online."
- **Question:** "Based on your SS1-SS3 knowledge, recommend and describe three (3) major digital technologies or systems that would be needed to build this 'School Portal'." (Justify *why* each is needed for the Principal's specific goals).

SS3 DIGITAL TECHNOLOGIES LESSON NOTES

THIRD TERM (FINAL REVISION & PROJECTS)

WEEK 1: COMPREHENSIVE REVISION OF SS1–SS3

1.1 INTRODUCTION

Welcome to your final term. This term is not about learning new topics, but about **consolidation** and **application**. We will synthesize everything from SS1 (Basic Hardware), SS2 (Programming/Web/Database), and SS3 (Advanced Systems) to prepare you for your practical projects and your final SSCE examination (WAEC/NECO).

1.2 THE FIVE PILLARS OF DIGITAL TECHNOLOGIES (COMPREHENSIVE REVIEW)

This is a high-level overview of the major areas you must know.

PILLAR 1: COMPUTER HARDWARE & TROUBLESHOOTING

- **SS1 Core Components:**
  - **CPU (Central Processing Unit):** The "brain." Executes all instructions.
  - **Motherboard:** The "nervous system." Connects all components.
  - **RAM (Random Access Memory):** *Volatile* (temporary) memory for *active* programs (your "work desk").
  - **PSU (Power Supply Unit):** Converts AC (from wall) to DC (for components).
  - **Storage (Non-Volatile):** Permanent memory (your "filing cabinet").
- **SS3 Storage (Advanced):**
  - **HDD (Hard Disk Drive):** Mechanical, spinning platters. Slow, cheap, high capacity.
  - **SSD (Solid State Drive):** Flash memory, no moving parts. Very fast, durable, more expensive. (SATA & M.2/NVMe form factors).
- **SS3 Expansion Components:**

- **GPU (Graphics Processing Unit):** A dedicated card for rendering images (gaming, video editing). Plugs into the **PCIe x16** slot.
- **iGPU (Integrated):** GPU built into the CPU. Weak, for basic tasks.

- **SS3 Troubleshooting:**
  - **POST (Power-On Self-Test):** The *first* check the BIOS/UEFI runs.
  - **Beep Codes:** How the motherboard reports a *hardware* failure *before* video starts. (e.g., 1 long, 3 short = Video Card; 3 short = RAM).
  - **BSOD (Blue Screen):** A *software* (OS or Driver) or *hardware* (RAM) crash. Check the "Stop Code."
  - **Boot Failure ("No Bootable Device"):** Check BIOS boot order, check SATA cables, or OS is corrupt.

PILLAR 2: PROGRAMMING & SOFTWARE DEVELOPMENT

- **SS2 Core Logic:**
  - **Variables:** Named containers for data (String, Integer, Boolean).
  - **Conditionals:** IF...ELSE (making decisions).
  - **Loops:** FOR...WHILE (repeating tasks).
- **SS3 Advanced Programming:**
  - **Functions:** Reusable, named blocks of code (DRY Principle: "Don't Repeat Yourself").
  - **Arrays/Lists:** A single variable that holds a collection of items.
  - **OOP (Object-Oriented Programming):**
    - **Class:** The "blueprint" (e.g., class Student).
    - **Object:** The "instance" built from the blueprint (e.g., student_Bello).
    - **Attributes:** Properties (e.g., student_Bello.name).
    - **Methods:** Actions (e.g., student_Bello.introduce_self()).
- **SS3 SDLC (Software Development Life Cycle):**
  - The 6 phases: **Planning -> Design -> Implementation (Code) -> Testing -> Deployment -> Maintenance.**
  - **Waterfall Model:** Linear, sequential (Finish Phase 1 before Phase 2). Inflexible.
  - **Agile Model:** Iterative (small "sprints"). Flexible, welcomes change.

PILLAR 3: NETWORKING & SECURITY

- **SS2 Networking:**
  - **Types:** LAN (Local/Office), WAN (Wide/Internet), MAN (Metro).
  - **Topologies:** Star (modern), Bus (old), Ring.
- **SS3 Advanced Networking:**
  - **IP Addressing:** IPv4 (e.g., 192.168.1.10) and IPv6.
  - **Subnet Mask:** (e.g., 255.255.255.0). Separates the Network ID from the Host ID.
  - **Protocols (The "Rules"):**
    - **TCP:** *Reliable* (checks for errors, e.g., file download).
    - **UDP:** *Fast* (no error check, e.g., video streaming).
    - **HTTP/HTTPS:** For websites (S = Secure).
    - **DHCP:** Automatically assigns IP addresses.
- **SS3 Network Security:**
  - **CIA Triad:** Confidentiality (secrecy), Integrity (trustworthy data), Availability (it works).
  - **Malware:** Virus, Worm (self-spreads), Trojan (disguised), Ransomware (encrypts).
  - **Firewall:** The "gatekeeper" that blocks/allows traffic.
  - **VPN (Virtual Private Network):** An "encrypted tunnel" for privacy and security.

PILLAR 4: WEB & DATABASE

- **SS2 Basic Web:**
  - **HTML:** The *structure* (skeleton).
  - **CSS:** The *style* (clothes).
- **SS3 Advanced Web:**
  - **JavaScript (JS):** The *behaviour* (muscles/brain). Makes the site interactive.
  - **DOM (Document Object Model):** The "map" of the HTML that JS manipulates.
  - **Responsive Design:** Using **CSS Media Queries** to make a site look good on both phones and desktops.
- **SS3 Database (SQL):**

- **RDBMS:** Relational Database Management System (e.g., MySQL).
- **SQL (Structured Query Language):** The language used to talk to a database.
- **DML:** SELECT (get data), INSERT (add data), UPDATE (change data), DELETE (remove data).
- **DDL:** CREATE TABLE (builds the structure).

PILLAR 5: MODERN TECHNOLOGIES & SYSTEMS

- **SS3 Virtualization & Cloud:**
  - **Virtualization:** A *hypervisor* (e.g., VirtualBox) runs a "virtual" PC (a VM) inside a physical one.
  - **Cloud Computing:** Renting IT resources (servers, storage) over the internet.
  - **IaaS:** Renting raw servers/storage (for IT admins).
  - **PaaS:** Renting a platform *to build* apps (for developers).
  - **SaaS:** Renting a *finished app* (e.g., Gmail, Netflix) (for end-users).
- **SS3 Internet of Things (IoT):** "Smart" devices (sensors) that connect to the internet to send/receive data (e.g., Smart Watch, Smart Home).
- **SS3 Data Protection:**
  - **Backup vs. Redundancy:** Backup = Spare tire (recovery). Redundancy = Dual-wheel truck (no downtime).
  - **3-2-1 Rule: 3** Copies, on **2** different media, with **1** copy off-site.
  - **RPO (Recovery Point Objective):** How much *data* can we lose?
  - **RTO (Recovery Time Objective):** How much *time* can we be down?
- **SS3 Project Management:**
  - **Iron Triangle: Time, Cost, Scope**. You can't change one without affecting the others.
  - **Tools: Kanban** (visual flow), **Gantt** (timeline/schedule).

EVALUATION (WEEK 1)

1. Explain the "3-2-1" rule of data backup.
2. Differentiate between the Agile and Waterfall SDLC models.

3. A client's PC gives "1 long, 3 short" beeps. What is the *most likely* hardware fault?
4. In OOP, what is the difference between a "Class" and an "Object"?
5. What is the purpose of a CSS Media Query?

ASSIGNMENT (WEEK 1)

1. Create a "Concept Map" (a diagram with bubbles and arrows) that links the following 10 terms together: Cloud (SaaS), IoT, JavaScript, HTML, SQL, Database, Responsive Design, Network, Firewall, VPN.
2. Your map should visually explain how these technologies interact to create a modern application.

WEEK 3: PRACTICAL PROJECT I (WEBSITE/DATABASE)

3.1 PROJECT BRIEF: "MY SCHOOL PORTAL"

Objective: To design the *prototype* for a simple, functional school portal. This project will combine your skills in web design (HTML/CSS/JS) and database management (SQL).

Project Components:

1. **The Database (The "Backend"):** The database (e.g., MySQL or SQLite) will store the information.
2. **The Website (The "Frontend"):** The HTML/CSS pages that the user interacts with.
3. **The Logic (The "Brain"):** The JS or server-side code that connects the frontend to the backend.

3.2 PART 1: THE DATABASE DESIGN

First, we must design the "schema" for our database. We will need two tables.

Table 1: Students

- This table holds the student's login and personal info.
- **SQL CREATE TABLE Command:**
- CREATE TABLE Students (
- StudentID VARCHAR(10) PRIMARY KEY,
- Password VARCHAR(255) NOT NULL,
- FirstName VARCHAR(100),
- LastName VARCHAR(100),
- Class VARCHAR(10)
-);

Table 2: Results

- This table holds the grades, linked by the StudentID.

- **SQL CREATE TABLE Command:**
- CREATE TABLE Results (
- ResultID INT AUTO_INCREMENT PRIMARY KEY,
- StudentID VARCHAR(10),
- Subject VARCHAR(50),
- TestScore INT,
- ExamScore INT,
- FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
-);

3.3 PART 2: THE WEBSITE (FRONTEND)

You will create two HTML files.

File 1: login.html (The Login Page)

- **HTML:** Needs a simple form with:
  - A text input for StudentID.
  - A password input for Password.
  - A "Login" button.
- **CSS:** Style the form to be centered, clean, and professional.

File 2: portal.html (The Dashboard)

- This is the page the user sees *after* logging in.
- **HTML:**
  - A "Welcome, [Student Name]" section.
  - A section for "My Details" (FirstName, LastName, Class).
  - A large table with headings: Subject, Test Score, Exam Score.
- **JavaScript:**
  - You will "mock" the login.
  - Write a JS function for the "Login" button.
  - If StudentID = "SS3-001" and Password = "1234", then redirect to portal.html.

- In portal.html, write JS to *dynamically* fill in the table with "dummy" data (e.g., "Maths - 30 - 55"). This simulates fetching data from the database.

3.4 DELIVERABLES (WHAT TO SUBMIT)

1. A text file (schema.sql) containing your two CREATE TABLE commands.
2. Your login.html file (with HTML, CSS, and JS).
3. Your portal.html file (with HTML, CSS, and JS).
4. (Optional) A simple "Home Page" (index.html) that links to the login page.
5. A 1-page report explaining how your system *would* work in a real-world scenario (e.g., how the JS would use fetch() to call a server, which would then run the SQL SELECT query).

EVALUATION (WEEK 3)

1. Complete the database design (schema.sql).
2. What is a PRIMARY KEY in SQL?
3. What is a FOREIGN KEY, and why is it used in the Results table?

ASSIGNMENT (WEEK 3)

1. Begin building the login.html and portal.html files. Focus on making them clean and responsive (use Media Queries!).
2. (Bonus) Add a simple JavaScript logout() function to the portal.html page that redirects the user back to login.html.

WEEK 6: CYBERSECURITY III

TOPIC: ETHICAL HACKING AND PENETRATION TESTING

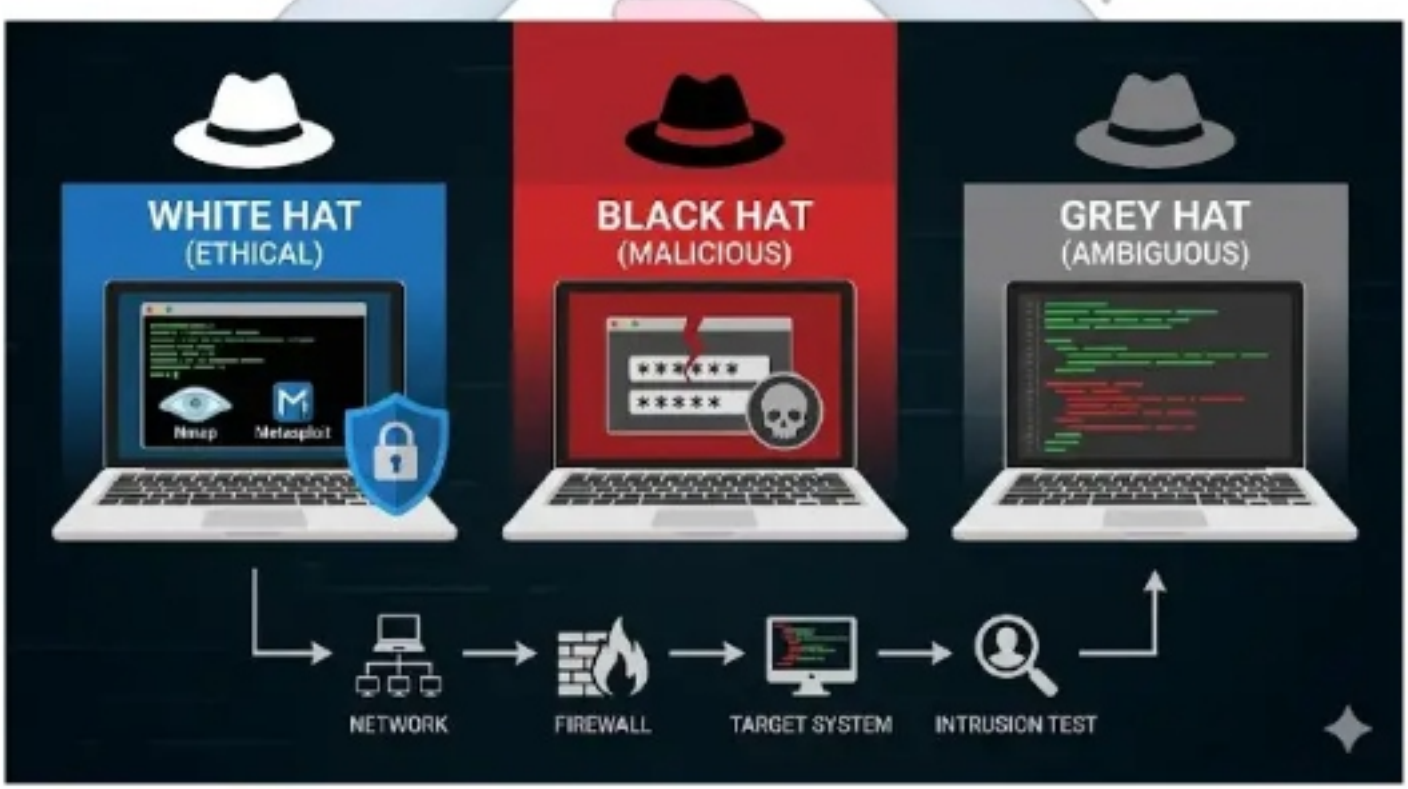
INTRODUCTION: We have previously discussed cybersecurity in terms of *defense* (passwords, firewalls). This week, we learn about *proactive defense*, also known as "offensive security." The best way to protect a system is to *think like a hacker* and find your own weaknesses before a criminal does. This is the job of an "Ethical Hacker."

1: THE "HACKER HATS" (TYPES OF HACKERS)

- **1. WHITE HAT HACKER:**
  - **THE "GOOD GUY."** Also known as an "Ethical Hacker" or "Security Researcher."
  - **MOTIVATION:** To help companies.
  - **METHOD:** Hacks with *explicit, written permission* from a company to find vulnerabilities. Their goal is to *fix* security holes.
  - **LEGALITY:** 100% legal. This is a high-paying, in-demand job.
- **2. BLACK HAT HACKER:**

- **THE "CRIMINAL."** Also known as a "Malicious Hacker."
- **MOTIVATION:** Personal gain (money), revenge, espionage, or pure destruction.
- **METHOD:** Hacks *without permission*. Steals data (credit cards, PII), deploys ransomware, or destroys systems.
- **LEGALITY:** 100% illegal. This is a crime.

- **3. GREY HAT HACKER:**
  - **THE "IN-BETWEEN."**
  - **MOTIVATION:** Ambiguous. Often hacks for fun, to show off, or for "vigilante" justice.
  - **METHOD:** Hacks *without permission* (like a Black Hat), but *may* report the vulnerability to the company (like a White Hat). They might also demand a "bug bounty" (payment) to fix it, which borders on extortion.
  - **LEGALITY:** Illegal. Accessing a system without permission is a crime,



regardless of intent.

2: WHAT IS PENETRATION TESTING (PEN-TESTING)?

- **DEFINITION:** A Penetration Test (or "Pen-Test") is the *formal, authorized, and simulated* cyberattack on a computer system, network, or application to evaluate its security.
- **GOAL:** To find and exploit vulnerabilities in a safe and controlled manner, *before* a Black Hat hacker can.
- **THE DELIVERABLE:** The final "product" of a pen-test is not the "hack"; it is a **detailed report** that explains:
    1. What vulnerabilities were found.
    2. How they were exploited (step-by-step).
    3. The *risk* to the business (e.g., "We were able to steal all customer data").
    4. The *recommendations* for fixing the vulnerabilities.

3: THE 5 PHASES OF A PENETRATION TEST

1. **1. RECONNAISSANCE (INFORMATION GATHERING):**
    - "Casing the joint." The hacker gathers as much *public* information as possible about the target (e.g., employee names from LinkedIn, technologies used, IP addresses).
2. **2. SCANNING:**
    - The hacker actively "touches" the target's network.
    - Uses tools (like Nmap) to scan for open "ports" (doors), what services are running (e.g., web server, database), and what software versions they are using.
3. **3. GAINING ACCESS (EXPLOITATION):**
    - The "hack." The ethical hacker uses a known vulnerability (e.g., an unpatched server, a weak password) or a "payload" to get inside the system.
4. **4. MAINTAINING ACCESS:**
    - The hacker tries to stay in the system, "pivot" to other systems on the network, and "escalate privileges" (e.g., moving from a normal user to a "root" or "admin" user).
5. **5. COVERING TRACKS & REPORTING:**
    - A Black Hat would delete logs to hide their presence.

- A White Hat *securely documents* every step and *writes the final report* for the client.

4: COMMON ATTACK VECTORS

- **SOCIAL ENGINEERING:** Hacking the *human*. This is the *most common* attack vector.
  - **PHISHING:** A mass e-mail campaign that *looks* legitimate (e.g., from your bank) to trick you into clicking a malicious link or entering your password.
  - **SPEAR PHISHING:** A highly *targeted* phishing attack on one specific person (e.g., the CFO).
  - **PRETEXTING:** A phone call where the hacker pretends to be someone else (e.g., "Hello, this is IT, I need your password to apply a security patch").
- **TECHNICAL ATTACKS:**
  - **UNPATCHED SOFTWARE:** Running old, outdated software with known security holes. This is the *most common* technical vector.
  - **SQL INJECTION (SQLI):** An attack that targets a website's database by "injecting" malicious code into a form (like a search bar).
  - **CROSS-SITE SCRIPTING (XSS):** An attack that injects malicious code into a website, which then runs in the *victim's* browser.

EVALUATION

1. Explain the key difference between a "White Hat," "Black Hat," and "Grey Hat" hacker.
2. What is "Penetration Testing," and what is its primary goal?
3. What is the *most important* final "product" that a White Hat hacker delivers to their client?
4. List the 5 phases of a penetration test.
5. Define "Social Engineering" and give one clear example (like phishing).

ASSIGNMENT

1. **RESEARCH:** What is a "Bug Bounty" program? How do companies like Google and Apple use them to improve their security?