STUDY MATERIAL OF CLASS 8

1. Understanding Algorithms and Decisions

Algorithms are step-by-step procedures or formulas for solving problems. In computer science, they are the foundation of all programs, whether it's sorting data, searching for items, or making decisions. Decision-making in programming involves logic that allows software to respond differently under different conditions—often implemented using conditional statements or machine learning.

1.1 What is an Algorithm?

An **algorithm** is a finite, well-defined sequence of steps or instructions designed to perform a specific task or solve a problem.

Example: A recipe for baking a cake is an algorithm – it includes steps like preheating the oven, mixing ingredients, and baking for a specific time.

In computer science, algorithms are the backbone of programming. Every task—be it searching, sorting, or calculating—is carried out by following an algorithm.

1.2 Key Characteristics of a Good Algorithm

1. **Input** – Clearly defined input(s).
2. **Output** – Clearly defined output(s).
3. **Definiteness** – Each step is precisely defined.
4. **Finiteness** – The algorithm must end after a finite number of steps.
5. **Effectiveness** – All operations must be basic enough to be carried out.

1.3 Types of Algorithms

Here are a few common algorithm types:

Algorithm Type	Description	Example
Search Algorithm	Finds an item in a dataset	Linear Search, Binary Search
Sort Algorithm	Arranges data in a particular order	Bubble Sort, Merge Sort
Recursive Algorithm	Calls itself to solve subproblems	Factorial, Fibonacci
Greedy Algorithm	Makes the optimal choice at each step	Dijkstra’s algorithm
Divide and Conquer	Breaks the problem into smaller sub-problems	Merge Sort, Quick Sort
Dynamic Programming	Solves complex problems by breaking into simpler overlapping subproblems	Knapsack problem, Fibonacci

1.4 What is Decision Making in Programming?

Decision-making allows a program to respond differently based on conditions or user inputs. This is implemented using **conditional statements** (if, else, switch) and sometimes **loops**.

✅ **Basic Decision-Making Structure:**

```
if condition:
    # do something
elif another_condition:
    # do something else
else:
    # fallback case
```

This enables dynamic responses in programs. For example, in an ATM system:

- If the pin is correct → show account balance
- Else → show error message

1.5 Algorithms in Real-Life Decisions

Algorithms don’t just live in code—they affect many real-world decisions:

- **Google Search** uses PageRank algorithm.

- **Netflix** uses recommendation algorithms.
- **Banking apps** use fraud detection algorithms.
- **Self-driving cars** use real-time decision-making algorithms for navigation and safety.

1.6 Importance in AI and Machine Learning

In **Artificial Intelligence (AI)** and **Machine Learning (ML)**, algorithms are used to:

- Train models.
- Optimize outcomes.
- Make predictions or classifications (e.g., spam filter, disease diagnosis).

Each decision in ML is based on a trained algorithm, such as:

- **Decision Trees**
- **K-Nearest Neighbors (KNN)**
- **Support Vector Machines (SVM)**
- **Neural Networks**

1.7 Why Learn Algorithms and Decisions Early?

Understanding how decisions are made and problems are solved:

- Builds problem-solving skills.
- Prepares students for advanced programming.
- Forms the base of software development, data analysis, robotics, cybersecurity, and AI.

2. INTRODUCTION TO PYTHON USING BLOCKS

Python is a beginner-friendly, high-level programming language widely used in data science, web development, and automation. Block-based coding platforms like **Google Colaboratory (Colab)** or **Blockly** make Python even more accessible to beginners.

2.1 Basics

Includes syntax, indentation (critical in Python), comments (#), and basic print() functions.

2.2 Variables, Constants, Keywords

- **Variables** are containers for storing data values. Example: x = 5
- **Constants** are fixed values that do not change during execution (though Python doesn't have true constants, by convention they are uppercase: PI = 3.14)
- **Keywords** are reserved words like if, else, for, while, def, which cannot be used as variable names.

Here is a **table of Python keywords** as of Python 3.11 (the most widely used versions). These are **reserved words** in Python — you **cannot use them as variable names**, function names, or identifiers.

Python Keyword Table

Keyword	Description
False	Boolean value representing false
None	Represents the absence of a value
True	Boolean value representing true
and	Logical AND operator
as	Used to create an alias (e.g., import as)
assert	Debugging aid that tests a condition
async	Used for asynchronous programming
await	Pauses async function execution until a task completes
break	Exits the current loop

Keyword	Description
class	Defines a class
continue	Skips the rest of the loop iteration
def	Defines a function
del	Deletes an object or variable
elif	Else if condition
else	Else block for conditional statements
except	Catches exceptions
finally	Executes code regardless of exceptions
for	Looping over a sequence
from	Specifies the module to import from
global	Declares a global variable
if	Conditional execution
import	Imports a module
in	Membership test
is	Tests for object identity
lambda	Creates an anonymous function
nonlocal	Refers to a variable in the nearest enclosing scope
not	Logical NOT operator
or	Logical OR operator
pass	Null operation; does nothing
raise	Raises an exception
return	Exits a function and returns a value
try	Defines a block of code to test for errors
while	Loops while a condition is true
with	Context manager (e.g., file handling)

Keyword	Description
yield	Pauses a generator and returns a value
match	Structural pattern matching (added in Python 3.10)
case	Used with match statements for pattern matching (3.10+)

Python Code to List All Keywords

You can use the built-in keyword module to get all current keywords:

```
import keyword
print("List of Python Keywords:")
print(keyword.kwlist)
```

2.3 Loops

Used for repetitive tasks.

- **for loops**: Iterate over a sequence.
- **while loops**: Run as long as a condition is true.

◆ **Conditional Statements**

- if, elif, and else control decision-making in code.

  Example:

```
if x > 10:
    print("Greater than 10")
```

2.4 Array

In Python, arrays are often implemented using lists.

```
arr = [1, 2, 3, 4]
```

2.5 Working with Google Colaboratory

Colab is a cloud-based Python notebook environment by Google. It:

- Supports Python and libraries like NumPy, pandas, TensorFlow.
- Requires no setup.

- Allows collaborative coding like Google Docs.

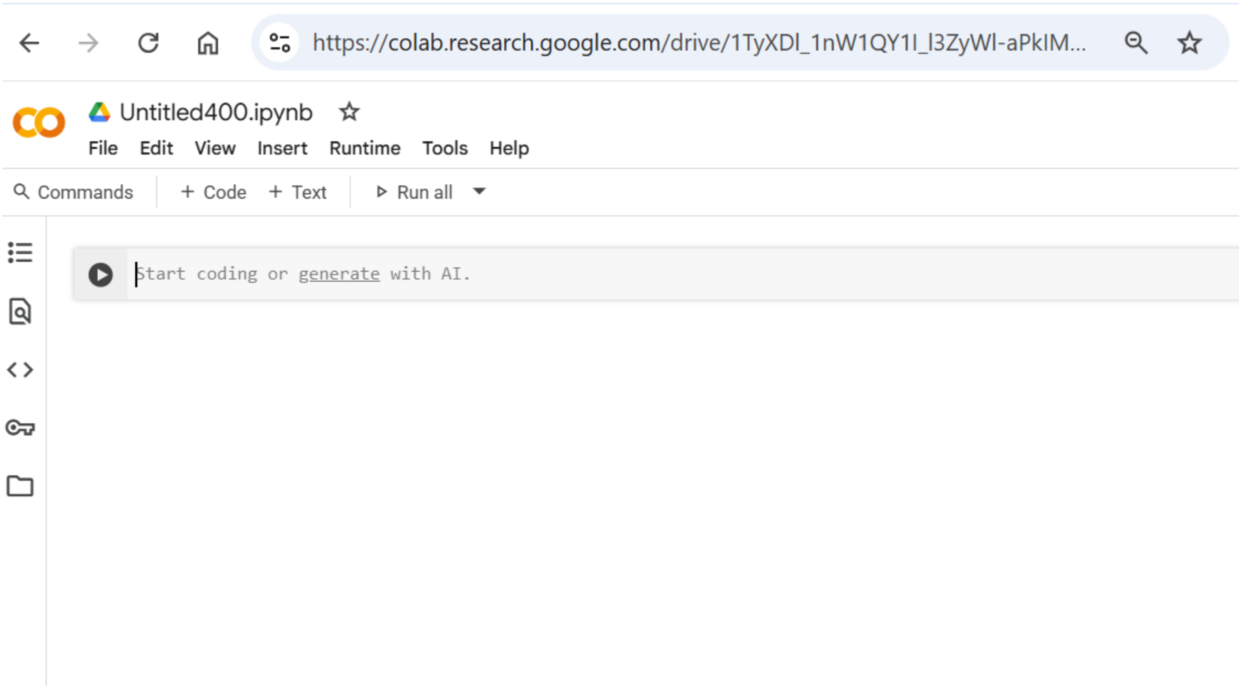

Fig 1 Google Collaboratory

2.6 Python Lab

Here's a structured Python lab manual introduction covering **basic concepts** such as **Introduction to Python**, **Data Types**, **Functions**, **Loops**, **If-Else**, and **Arrays**, ideal for a school or college-level curriculum:

🐍 Python Lab Manual – Basic Concepts

🧾 1. Introduction to Python

Python is a high-level, interpreted, general-purpose programming language known for its simplicity and readability. It is widely used in web development, data science, automation, machine learning, and more.

🔑 Key Features:

- Easy syntax (similar to English)
- Dynamically typed
- Interpreted language
- Open-source and large community support

- Extensive standard libraries

🛠 **First Python Program:**

```
print("Hello, World!")
```

🔤 2. Python Data Types

Python supports various **built-in data types**:

Data Type	Example	Description
int	10	Integer
float	3.14	Floating-point number
str	"Hi"	String
bool	True	Boolean (True/False)
list	[1, 2]	Ordered, mutable collection
tuple	(1, 2)	Ordered, immutable collection
dict	{"a":1}	Key-value pairs
set	{1, 2}	Unordered, no duplicate values

Example:

```
a = 10
b = 3.5
c = "Python"
d = True
print(type(a), type(b), type(c), type(d))
```

🔁 3. Loops

➤ **For Loop**

```
for i in range(5):
    print(i)
```

➤ While Loop

```
count = 0
while count < 5:
    print(count)
    count += 1
```

🔄 4. If-Else Statements

Used for decision making.

```
num = 7
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

🔧 5. Functions in Python

Functions help modularize and reuse code.

➤ Defining a Function

```
def greet(name):
    print("Hello, " + name)

greet("Alice")
```

➤ Function with Return

```
def add(a, b):
    return a + b

result = add(3, 5)
print(result)
```

📚 6. Arrays (Using List)

Python does not have built-in arrays like C/C++; it uses **lists** which are dynamic arrays.

```
arr = [10, 20, 30, 40]

# Access elements
print(arr[0])

# Update element
arr[1] = 25

# Append new element
arr.append(50)

# Loop through array
for item in arr:
    print(item)
```

✅ Sample Lab Exercise

Task 1:

- Create a function to calculate the factorial of a number using a loop.

Task 2:

- Write a program to take a list of numbers, and print all even numbers using for loop.

Task 3:

- Write an if-else program to check if a number is odd or even.

3. What Are Digital Signatures?

Digital signatures are cryptographic methods of verifying the authenticity and integrity of digital messages or documents. They ensure:

- **Authentication** (verifying sender),
- **Integrity** (data not altered),
- **Non-repudiation** (sender cannot deny sending it).

They rely on **asymmetric encryption** (public-private key pairs).

4. What Is a Hard Wallet?

A **hardware wallet** (or cold wallet) is a physical device that securely stores a user's private keys offline. It:

- Provides high security for cryptocurrencies.
- Is immune to online hacks.
- Examples: Ledger Nano S, Trezor.

5. What Is a Soft Wallet?

A **software wallet** (or hot wallet) is an application or software that stores private keys on internet-connected devices like phones or PCs.

- Easier to access and transact.
- Examples: MetaMask, Trust Wallet.
- More vulnerable to cyber attacks than hard wallets.

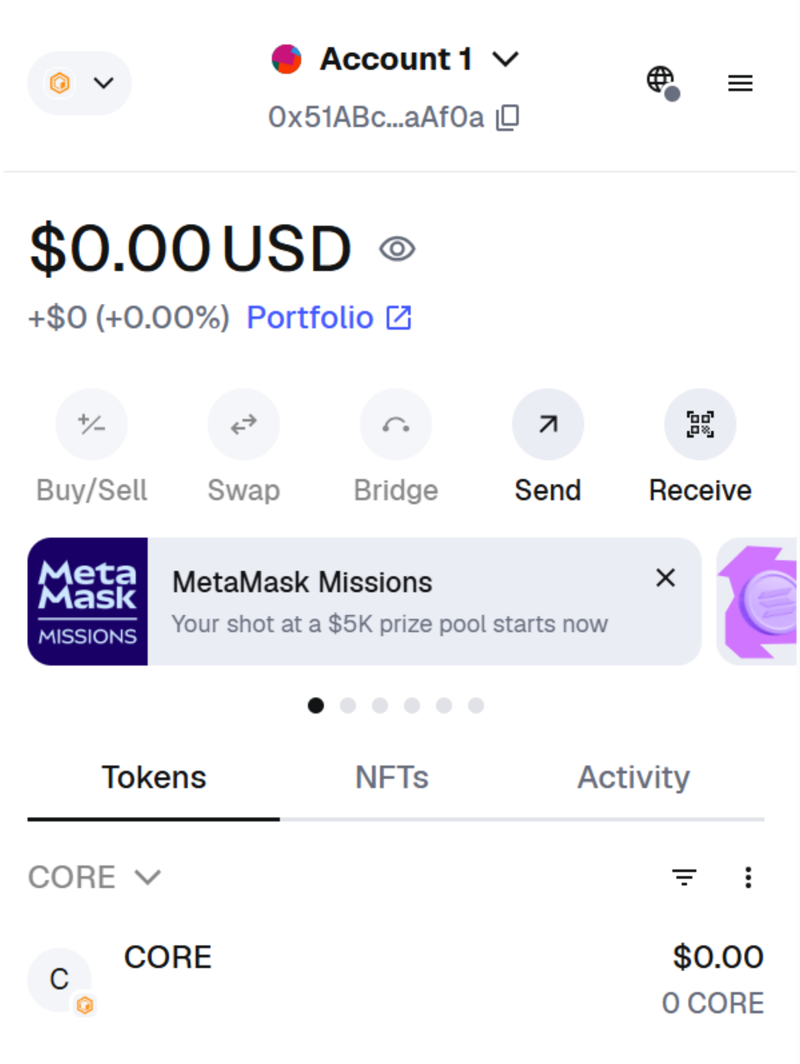

Fig 2 Metamask Wallet

Metamask Configuration

Here's a **step-by-step process for configuring MetaMask**, whether you're using the browser extension or mobile app:

🔧 1. Install MetaMask

✅ **For Browser (Chrome, Firefox, Brave, Edge):**

- Visit https://metamask.io
- Click **"Download"**
- Choose your browser and install the MetaMask extension.
- Pin the MetaMask icon for easy access (optional but helpful).

✅ **For Mobile:**

- Download from **Google Play Store** or **Apple App Store**
- Search for "MetaMask" by **Consensys** (ensure it's the official app)

🔐 2. Create a New Wallet

- Click **"Get Started"**
- Select **"Create a Wallet"**
- Agree or decline the data-sharing option
- Create a **strong password**
- Click **"Create"**

🔑 3. Secure Your Secret Recovery Phrase

- MetaMask will show a **12-word Secret Recovery Phrase**
- **Write it down** and store it **offline** (never share this with anyone!)
- Confirm the phrase in the correct order when prompted

🌐 4. Add a Network (Optional for Non-Ethereum Chains)

MetaMask defaults to Ethereum Mainnet. To interact with other networks (like Binance Smart Chain, Core Chain, Polygon, etc.):

▼ **Steps to Add a Network:**

- Click the network dropdown (usually says "Ethereum Mainnet")
- Select **"Add network"**
- Enter the network details manually (e.g., for Binance Smart Chain):

Field	Value
Network Name	Binance Smart Chain
RPC URL	https://bsc-dataseed.binance.org/
Chain ID	56
Currency	BNB
Block Explorer URL	https://bscscan.com

Click **Save**

📥 5. Import Existing Wallet (Optional)

If you already have a wallet:

- Click **“Import Wallet”**
- Enter your **Secret Recovery Phrase**
- Set a new password

💸 6. Fund Your Wallet

You can fund MetaMask with cryptocurrency by:

- Copying your wallet address (starts with 0x...)
- Sending crypto from an exchange (like Binance, Coinbase) or another wallet

⚙ 7. Advanced Configuration (Optional)

- **Custom Tokens**: To view a token not shown by default:
  - Click **“Import Tokens”**
  - Enter contract address, token symbol, and decimals
- **Connected Sites**:

- MetaMask asks for permission before connecting to dApps (DeFi, NFT platforms)
- You can view/manage connected sites under **Settings > Security & Privacy > Connected Sites**

You can now use MetaMask to:

- Send/receive crypto
- Connect to decentralized apps (dApps)
- Trade on DEXs like Uniswap, PancakeSwap
- Mint NFTs
- Use DeFi platforms like Aave, Compound, etc.

6. WEB 1.0 / WEB 2.0 / WEB 3.0 – EVOLUTION OF THE INTERNET

🌐 Web 1.0 – The Static Web (1990–2004)

- Web 1.0 is the **first generation** of the internet.
- It featured **static pages** that could only be read.
- There was **no user interaction** or dynamic content.
- Users were consumers, not contributors.

Example: Personal websites, early news portals, online brochures.

🌐 Web 2.0 – The Social Web (2004–Present)

- Web 2.0 introduced **interactivity**, **social networking**, and **user-generated content**.
- Platforms allow users to **comment, upload, and share**.
- Dominated by big tech companies.

Example: Facebook, YouTube, Wikipedia, Instagram, Twitter.

🌐 Web 3.0 – The Decentralized Semantic Web (Emerging)

- Web 3.0 aims to make the web **smarter, decentralized, and more secure**.
- Integrates **blockchain**, **AI**, **machine learning**, and **semantic search**.
- Focuses on **user data ownership**, **privacy**, and **decentralized apps (dApps)**.

Example: Ethereum, IPFS, Uniswap, Icecreamswap, Archerswap, OpenSea, Brave Browser.

Comparison Table: Web 1.0 vs Web 2.0 vs Web 3.0

Feature	Web 1.0	Web 2.0	Web 3.0
Timeline	1990–2004	2004–Present	Emerging (2020s onward)
Nature	Static and Read-only	Interactive and Social	Intelligent and Decentralized
User Role	Consumers only	Consumers + Contributors	Owners + Participants
Content Type	Static HTML Pages	Dynamic, user-generated content	Linked data, AI-generated content
Technologies	HTML, HTTP	AJAX, JavaScript, APIs	Blockchain, AI, Semantic Web, dApps
Data Storage	Centralized servers	Cloud-based centralized databases	Decentralized networks (blockchain/IPFS)
Control	Website owners	Central companies (Big Tech)	Users (via smart contracts, DAOs)
Examples	GeoCities, Yahoo Directory	Facebook, YouTube, Twitter	Ethereum, Uniswap, OpenSea, Brave
Privacy	No privacy controls	Limited privacy, data exploitation	Enhanced privacy, user-controlled data
Monetization	Banner Ads	Ads, user data monetization	Token economy, crypto payments
Innovation Focus	Access to content	Sharing and interaction	Ownership, trust, AI understanding

---

✅ **Summary**

- **Web 1.0** = Read
- **Web 2.0** = Read + Write
- **Web 3.0** = Read + Write + Own

Version	Features
Web 1.0	Read-only web (static pages, limited interaction)
Web 2.0	Interactive and social web (blogs, social media, user-generated content)
Web 3.0	Decentralized, blockchain-powered, AI-integrated web (dApps, smart contracts, data ownership)

7. What Is a Smart Contract?

Smart contracts are self-executing programs stored on a blockchain that run when predefined conditions are met. They automate workflows and remove the need for intermediaries in transactions.

7.1 Token as Smart Contract

A **token** is a digital asset implemented using smart contracts. They:

- Represent currencies, loyalty points, or assets.
- Are programmable with logic for transfers, supply, and governance.

7.2 NFT as Smart Contract

NFTs (Non-Fungible Tokens) are unique digital assets (e.g., art, music, tickets) encoded using smart contracts (typically ERC-721 or ERC-1155). Each token is distinct and provably scarce, ensuring digital ownership.

8. COIN AND TOKEN

8.1 What Is a Coin?

- Coins are native to a blockchain.
- They are mainly used for payments, mining rewards, or fees.
- **Coins = Digital cash**

Examples:

- **Bitcoin (BTC)** → native coin of Bitcoin blockchain
- **Core** → Native coin of Core Blockchain
- **Ethereum (ETH)** → native coin of Ethereum blockchain
- **BNB** → native coin of BNB Chain

8.2 What Is a Token?

- Tokens are built using **smart contracts** on an existing blockchain.
- They can represent anything — **currency, assets, tickets, governance rights, or even artwork**.
- Tokens use standards like **ERC-20 (fungible)** or **ERC-721 (NFTs)**.

Examples:

- Coredaovip (corevip), 9nftmania (9nm), PremiumDomain (PD), ARS, CoreID (CID)
- **NFTs** → digital art or collectibles represented as ERC-721 tokens

8.3 Coin vs Token

Feature	Coin	Token
Definition	A cryptocurrency that runs on its **own blockchain**	A digital asset built **on top of another blockchain**
Blockchain	Has its **own native blockchain**	Uses an **existing blockchain**, like Ethereum
Examples	Bitcoin (BTC), Ethereum (ETH), Litecoin (LTC)	USDT (Tether), UNI, SHIBA, LINK (on Ethereum)
Use Case	Primarily used as **money or store of value**	Represents assets, access rights, voting, etc.
Transaction Fees	Paid using the same coin	Paid using the native coin of the host blockchain (e.g., ETH for ERC-20 tokens)
Created With	Requires developing a **full blockchain protocol**	Created using **smart contracts** on existing blockchains
Fungibility	Always **fungible** (each unit is the same)	Can be **fungible or non-fungible** (e.g., NFTs)

🔄 In Simple Terms:

◈ **Coin**: Native currency of its own blockchain (Core)

◈ **Token**: Built on an existing blockchain (like Corevip, Blackdoge)