PROGRAMMING CLUB

Documentation

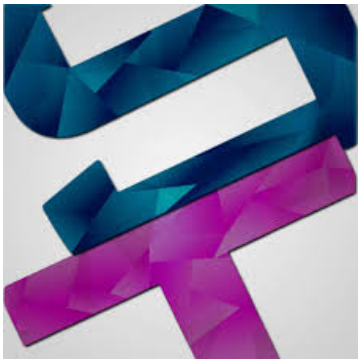
MASTERING WEB 3.0

Mentees

Somya Kumar–Soma Koushik–Ayush Rajput–Rahul Singh–Rushabh Pandya–Abhinav Pandey–Kaushey

Mentors

Chayan Kumawat
Tamoghna Kumar
Kinchit Goyal
Mayank Aggrawal



Summer Project - 2024

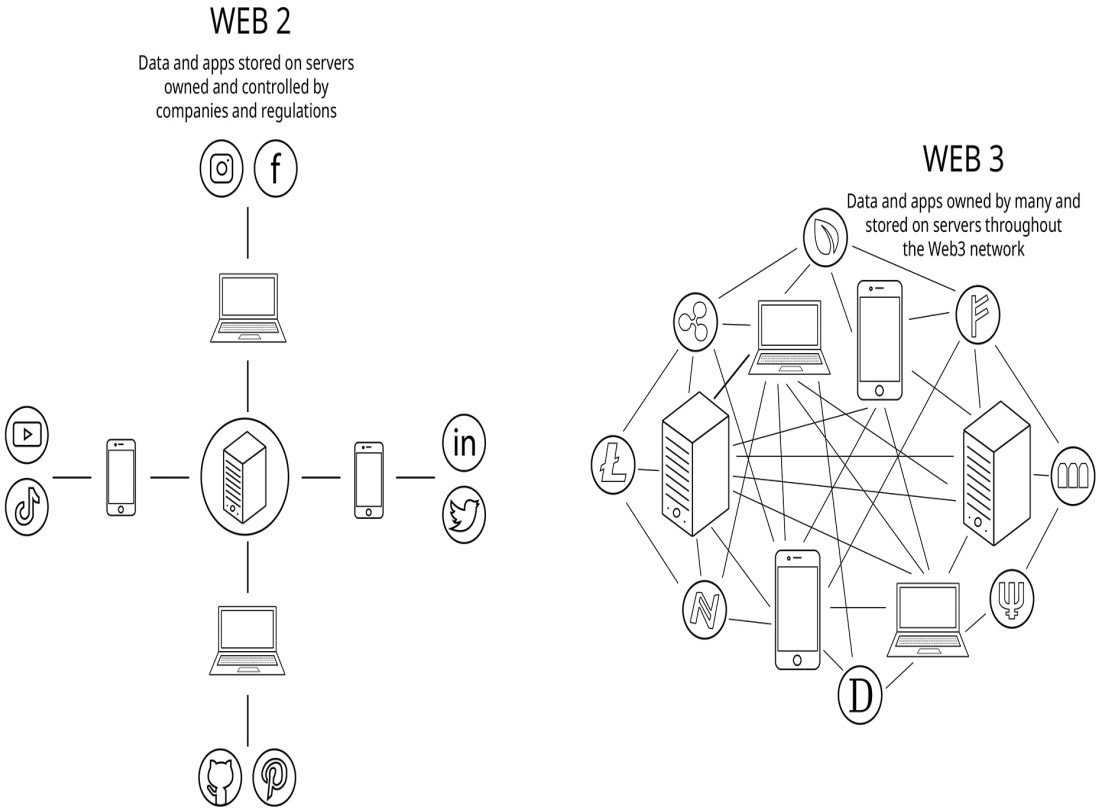
Project Timeline

Contents

1 Introduction to Web 3.0

Web 1.0 was all about fetching, and reading information.
Web 2.0 is all about reading, writing, creating, and interacting with the end user.
Web 3.0 is the third generation of the World Wide Web, and is a vision of a decentralized web which is currently a work in progress. It is all about reading, writing, and owning.
So, as usual Web 3.0 goes deeper into building a fairer and more transparent internet. For this reason, Web 3.0 is often associated with blockchain technology.

The first implementation of blockchain began with a man named **Satoshi Nakamoto**(anonymous), who invented Bitcoin and brought blockchain technology to the world back in 2009 after the 2007-8 Global Recession. The term "Web3" was coined by Polkadot founder and Ethereum co-founder Gavin Wood in 2014, referring to a "*decentralized online ecosystem based on blockchain.*" Ethereum, launched in 2015 by Vitalik Buterin and others, took blockchain technology beyond simple transactions via introducing the concept of smart contracts which enabled the creation of decentralized applications (D-Apps) that could execute programmable actions without relying on centralized authorities and luckily creating one of these apps is the final commitment of our project.



If you use any device that uses any blockchain technology (like Bitcoin) then your device is considered a **node** among many others which constitute the whole Web-3 network. The sole purpose of each node is to share data with other nodes and to store a copy of it within itself.(and to follow some specific rules while doing that. :P)

2 Blockchain Technology : BitCoin

Bitcoin is an electronic payment system that allows anyone to create an account and send any amount of money to anyone in the world.
Bitcoin solves the problem of being able to have a payment system that operates without a central point of control thus there is no need of that **trust** factor that existed in traditional banking system. Each node that is in the Bitcoin network has a copy of a file called blockchain, which is a big list of validated transactions.

It was possible to relay transactions across a network of computers before Bitcoin. However, the problem is that you can insert conflicting transactions in to a network of computers. For example, you could create two separate transactions that spend the same digital coin, and send both of these transactions in to the network at the same time.(**"Double-Spend"**)

2.1 Basic Terms

- **Transactions** : A bitcoin transaction contains information about the amount being sent, the account it is being sent from, and the account it is being sent to.You can think of a transaction as being part of a chain of outputs; one transaction creates an output, and then a future transaction selects that output (as an input) and unlocks it to create new outputs.

- **Blockchain** : A blockchain is a distributed ledger with growing lists of records (blocks) that are securely linked together via cryptographic hashes. It is called the "blockchain" because new transactions are added to the file in blocks, and these blocks are built on top of one another to create a chain of blocks. The blockchain is permanent storage for bitcoin transactions.(explore the chain !)

- **Block** : A block is a container for storing new transactions in the blockchain. Blocks are constructed during the process of mining. If you are a visual learner click here.

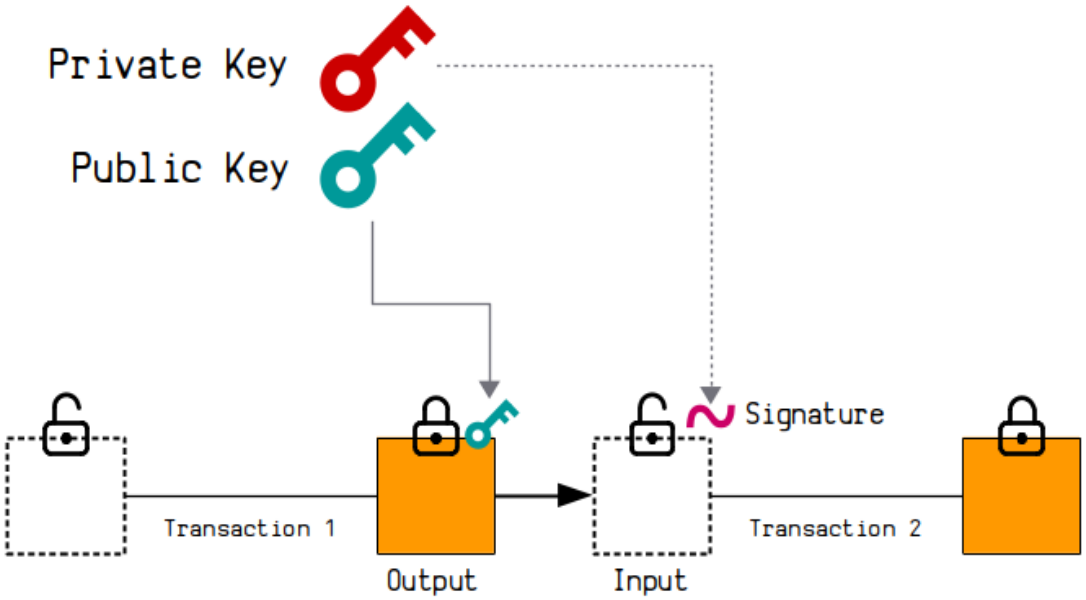
- **Mining** : Mining is the process of trying to add a new block of transactions on to the blockchain. It's a network-wide competition where any node on the network can work to add the next block on to the chain. New bitcoins are generated as a reward.

- **Mempool** : The memory-pool is the waiting area for transactions (temporary storage). New transactions are stored in a node's memory pool while they're waiting to get mined on to the blockchain.

2.2 Cryptography used in Bitcoin

- **Hash Functions** : A hash function is a programming tool that creates fingerprints for data. It takes in any amount of data, scrambles it, and returns a short and unique result for that data of specific size.If we are given a hash then its impossible to predict the input data of it. Eg- MD5,SHA-1,SHA-256, HASH-256 (double SHA-256) etc...
  Try it yourself

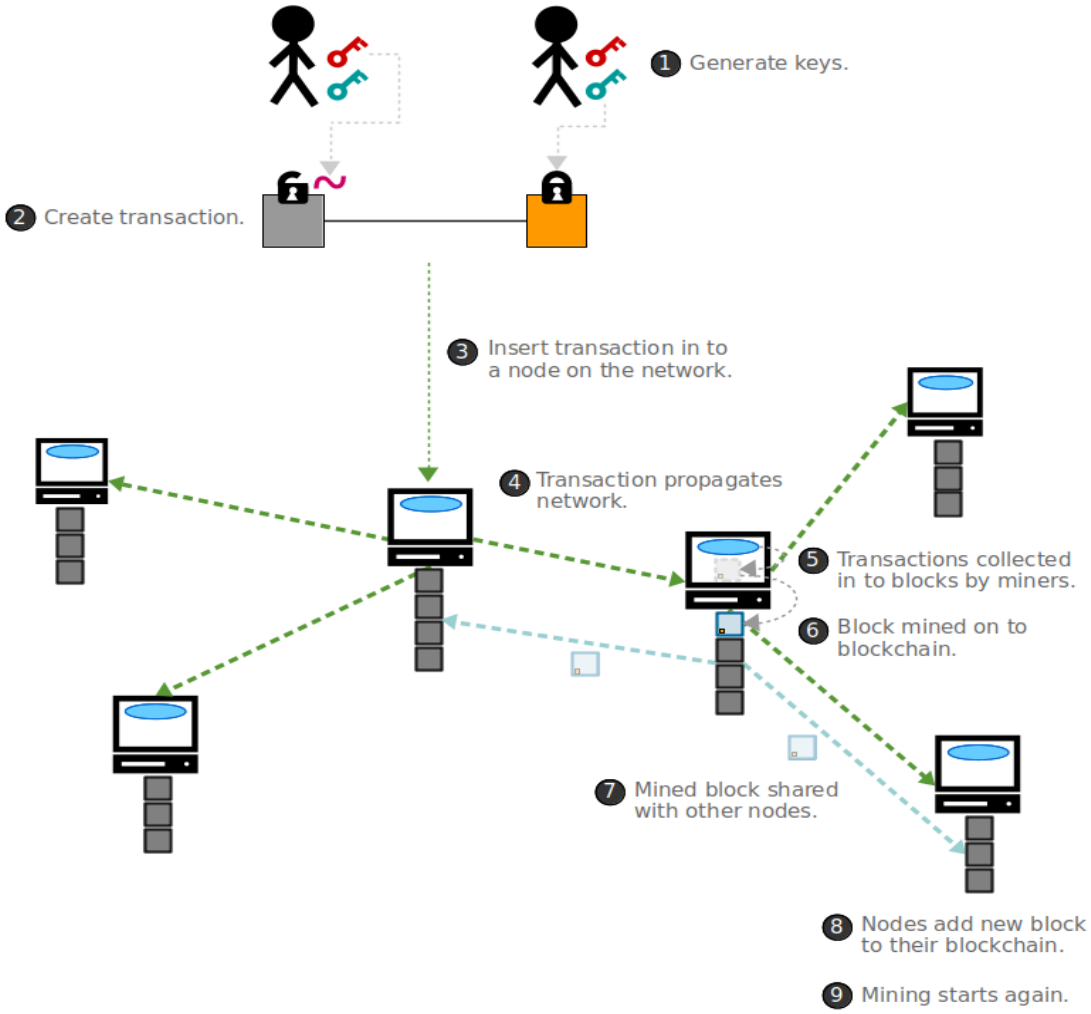
- **Public Key Cryptography** : Public key cryptography involves using mathematics to create a pair of keys: a private key and a public key. We can generate public key from private key but not vice-versa.Bitcoin utilizes this kind of public key cryptography as the basis for the locking mechanism inside transactions.A **digital signature** is created by using the private key to "sign" the entire transaction which is unique to that transaction and can't be reused.



2.3 Basic Working of Bitcoin

1. To use Bitcoin, you generate your own private key and public key. Your private key is a very large random number, and your public key is calculated from it.
2. To receive bitcoins, you give the person sending you them your public key. This person then creates a **transaction** where they unlock bitcoins that they own, and create a new "safe deposit box" of bitcoins and put your public key inside the lock.
3. This transaction is then sent to a node, where it's relayed from computer to computer until every node on the network has a copy of the transaction in their **mempool**. From here, each node has the opportunity to try and mine the latest transactions they have received on to the **blockchain**.
4. The process of **mining** involves a node collecting transactions from its memory pool in to a **block**, and repeatedly hashing the block to try and get a block hash below the current target value specified by the network.
5. The first miner to find a block hash below the target will add the block to their blockchain, and broadcast this block to the other nodes on the network. Each node will then verify and add this block to their blockchain (removing any conflicting transactions/ **Double-spend** from their mempool.), and restart the mining process to try and build on top of this new block in the chain.
6. Lastly, the miner who mined this block will have placed their own special transaction (coinbase) inside the block, which allows them to collect a set amount of bitcoins that did not already exist. This block reward acts as an incentive for nodes to continue to build the blockchain, whilst simultaneously distributing new coins across the bitcoin network.
   (Block reward = block subsidy(coinbase txn)+transaction fees for all txns).

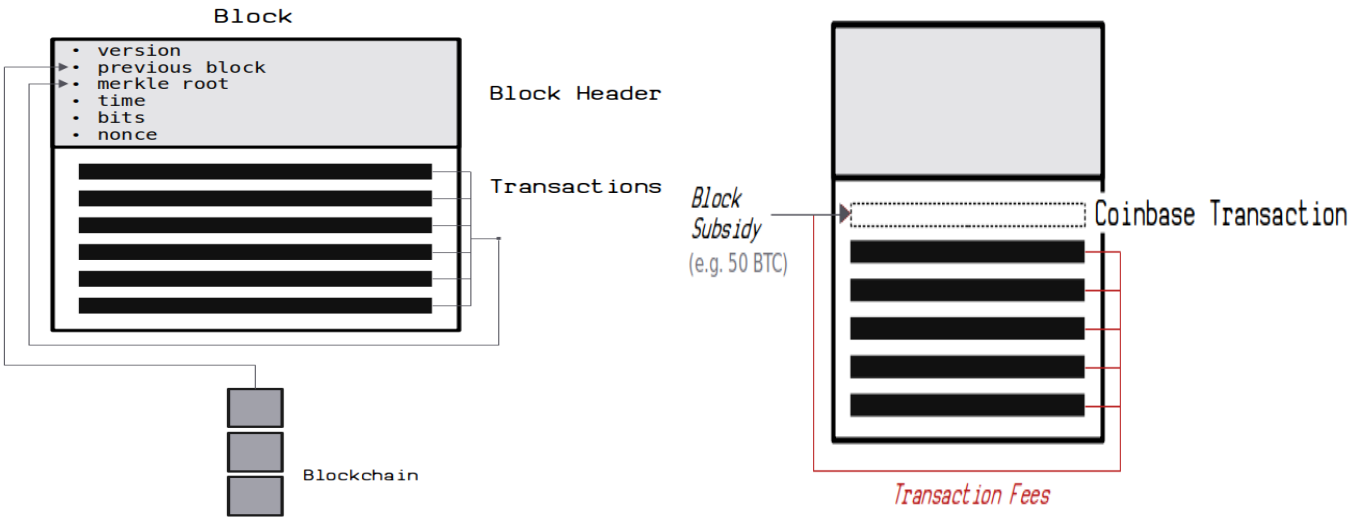
FINALLY THE LONGEST CHAIN IS ACCEPTED BY THE NETWORK



Note:- All the data of transactions stored on the blockchain is NOT in readable format (Hashed). We just know that the transaction occurred but don't know the bodies who facilitated the transaction.

2.4 Block Header

Each block has a **block header** which summarizes all of the data inside that block. This contains a fingerprint of all the transactions in the block, as well as a reference to a previous block. A random block header in Hex format:-
`0200000035f29a85a9b674a47fb2d5131667da1ca1c85ab06bfc2f30000000000000000bac8e1a6771a4c615997f9f12e94ad280f982536807caa9b4bf583111a55c2bdef2e9f539a855d186757ce1d`
It is 160 characters long that is **80 bytes**.
Its **Block Hash** is:
`00000000000000014e273f03fd7b705f7df4c674b8e351f8c49e41fe7dfc8ea`
(Its just HASH-256 of the block Header).There are example tools available **click here**



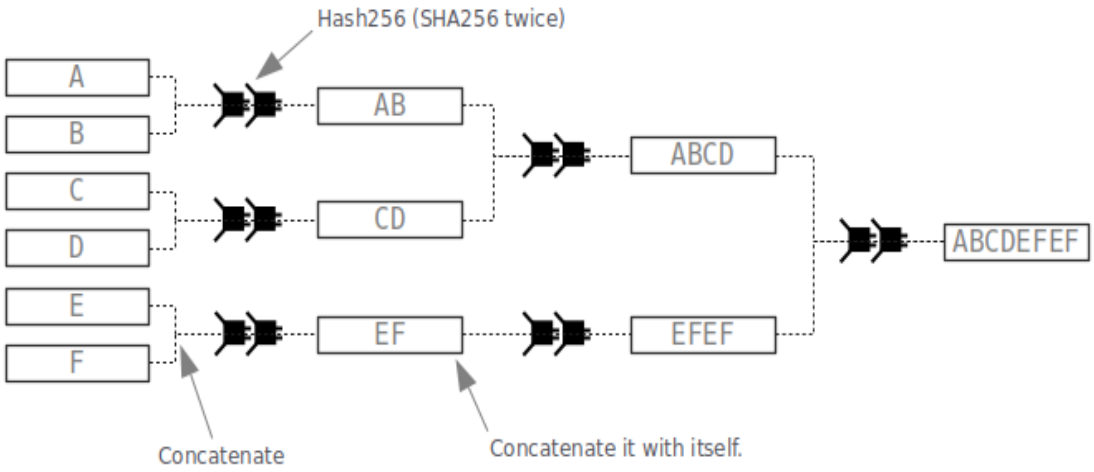
BLOCK HEADER STRUCTURE

Name	Type	Bytes	Description
version	int32_t	4	Version number
previous hash	char[32]	32	Hash of the previous block header in internal byte order
merkle root	char[32]	32	Merkle root of the transactions included in the block formatted in internal byte order
time	uint32_t	4	Epoch timestamp of the block
bits	uint32_t	4	Encodes the network target difficulty
nonce	uint32_t	4	Dedicated number to be updated to generate unique hashes

Merkle Root

A merkle root is created by hashing together pairs of TXIDs(transaction ID) to get a short and unique fingerprint for all the transactions in a block. We have to hash pairs until we get a single hash(if during a step we are left with a single item then hash with it itself).
The raw output is called merkle root(natural byte order) and the merkle root we see in header is in reverse byte order.
function to reverse byte order

```
def reverse_byte_order(hexstr):
REV = ''.join(reversed([hexstr[i:i+2] for i in range(0, len(hexstr), 2)]))
return REV
```

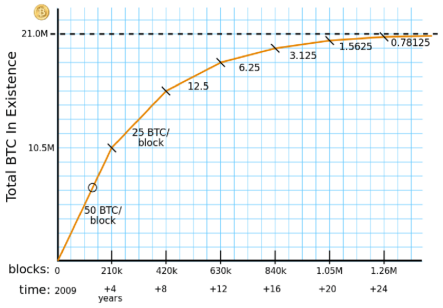


functions to calculate Merkle root

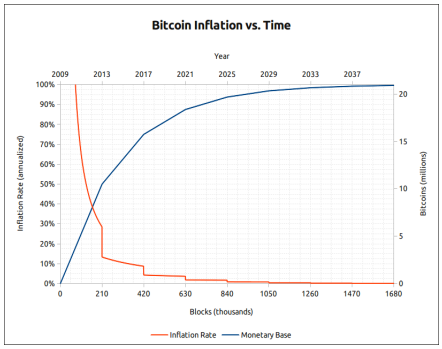
```
    def HASH_two(firstTxHash, secondTxHash): #Function to hash 2 consecutive txids
        unhex_reverse_first = binascii.unhexlify(firstTxHash)[::-1]
        #binascii.unhexlify() converts the txid in hex into bytes.
        unhex_reverse_second = binascii.unhexlify(secondTxHash)[::-1]
        #[:: -1] reverses the byte order to little-endian format (natural byte order)
        concat_inputs = unhex_reverse_first + unhex_reverse_second
        final_hash_inputs = hashlib.sha256(hashlib.sha256(concat_inputs).digest()).digest() # APPLY HASH256
        return binascii.hexlify(final_hash_inputs[::-1]) #Again reversed

    def merkleCalculator(transactions): #To calculate merkle root via recursion
        if len(transactions) == 1:
            return transactions[0]
        newHashList = []
        for i in range(0, len(transactions)-1, 2):
            newHashList.append(HASH_two(transactions[i], transactions[i+1]))
        if len(transactions) % 2 == 1: # odd, hash last item twice
            newHashList.append(HASH_two(transactions[-1], transactions[-1]))
        return merkleCalculator(newHashList)
```

2.5 Future of BitCoin

First of all, there are limited bitcoins that are mined or about to be mined which is 21 Million. After this limit no new bitcoins will be generated after mining and miners will only earn transaction fees. This limit maintains the value of this cryptocurrency as suggested by Mr Nakamoto. This limit will be reached(approx) in 2140 and after that it may be treated as a store of value as its quantity will be limited on Earth like gold.



29 more 1/2s needed after 2024 to reach limit



After every 4 years **inflation rate** dips, overall value increases

3 Consensus Mechanisms

Generally, consensus means that the majority of a group has agreed in favour of a decision. Consensus mechanisms ensure that all participants in the network agree on the validity of transactions and the state of the blockchain, thereby maintaining the integrity and security of the system without requiring a central authority. There are 2 famous mechanisms that are used in blockchain technology :-

- **Proof of Work (PoW)**
  The Proof of Work consensus algorithm involves solving a computationally challenging puzzle in order to create new blocks in the blockchain. The work done by the miner (who first solved the puzzle) is the proof which shows that the block is valid.
  Principle : *"A solution that is difficult to find but is easy to verify."*
  Bitcoin's PoW System :- The miners bundle up a group of transactions into a block and try to mine. To mine it, a hard mathematical problem has to be solved.
  Problem : *"Given data A, find a number x such as that the hash of x appended to A results is a number less than B."*
  where A is the **Block Hash** ,B is the **Target value** and x is **nonce**.
  Target value - A number that the header of a hashed block must be equal to or less than for that block to be added to the blockchain.
  Difficulty - The difficulty is a number that represents how difficult it is for miners to add new block to the blockchain. The difficulty ensures that blocks of transactions are added to the blockchain at regular intervals during mining, even as more miners join the network.(In case of bitcoin time interval is 10mins. and difficulty changes every 2weeks/2016 blocks).
  Difficulty is inversely proportional to the Target value.
  `basic iterative code to find nonce -`

```python
import hashlib
s = 'Mastering Web 3.0'
nonce = 0
target = int('0x0fffffff0000000000000000000000000000000000000000000000000000000', 16)
#int(string,16) converts hex(base 16) into integer for comparision
while True:
    value = s + str(nonce)
    hash_value = hashlib.sha256(value.encode('utf-8')).hexdigest() #SHA-256 function
    hash_int = int(hash_value, 16)
    if hash_int <= target:
        break
    nonce += 1
print("Nonce:", nonce)
print("Hash value:", hash_value)
```

  Click here to get visualisation about blockchain,nonce and other terms.

  Challenges -

  - **51% Attack**: If a controlling entity owns 51% of nodes or computing power then the node can control the entire mining system which is fatal.
  - **High Energy Consumption**: Miners consume high amounts of computing power in order to find the solution to the hard mathematical puzzle.
  - **Time-Consumption**: Miners have to iterate over many nonce values to find the right solution to the puzzle, which is a time-consuming process and transactions take 10mins. which is uneconomical.

- **Proof of Stake (PoS)**
  Nodes on a network stake an amount of cryptocurrency to become candidates to validate the new block and earn the fee from it. Then, an algorithm chooses from the pool of candidates the node which will validate the new block. This selection algorithm combines the quantity of stake with other factors:

- **Coin-age based selection**: The algorithm tracks the time every validator candidate node stays a validator. The older the node becomes, the higher the chances of it becoming the new validator.
- **Random Block selection**: The validator is chosen with a combination of 'lowest hash value' and 'highest stake'. The node having the best weighted-combination of these becomes the new validator.

Principle : *"Its based on staking i.e the value we bet on a certain outcome."*
PoS Mechanism:Proof-of-stake is a way to prove that validators have put something of value into the network that can be destroyed if they act dishonestly. The block validated by the selected validator is then checked by the other nodes. If they all agrees then the block added to the blockchain and transaction fee is rewarded to the validator with the stake othewise the stake is lost.
Features -

- **51% attack immunity**: A person attempting to attack a network will have to own 51% of the stakes(pretty expensive). This leads to a secure network. As there might not be much currency to buy such a share.
- **Energy-efficient**: As all the nodes are not competing against each other to attach a new block to the blockchain, energy is saved. Also, no problem has to be solved thus saving the energy. (more time efficient)

Challenges -

- **Centralization Risk**: In PoS, the likelihood of being chosen as a validator is proportional to the amount of cryptocurrency staked. This can lead to a concentration of power among wealthy participants who can afford to stake large amounts.
- **The 'Nothing at Stake' problem**: As there is no computational effort to validate blocks some nodes can support multiple blockchains in the event of a blockchain split(blockchain forking). In the worst-case scenario, every fork will lead to multiple blockchains and validators will work and the nodes in the network will never achieve consensus.

4 Bitcoin Mining Simulation : Assignment-1

In our Assignment we simulated the mining process of a block by validating and including transactions from a given set of JSON files in the mempool folder and saving block header in output.txt file(it includes a simple block header format, predefined coinbase transaction. and list of valid transaction IDs.) Assignment Link

---

```
import hashlib  # for hash algorithms
import json     # for json files
import os       # to link os files
import binascii # to convert to hex and other systems
import time
ut = time.time() # for unix epoch time

def SHA256(data):
    return hashlib.sha256(data.encode('utf-8')).hexdigest()

OUTPUT_FILE = 'output.txt'
Mempool_folder ='mempool'

def reverse_byte_order(hexstr):
def HASH_two(firstTxHash, secondTxHash):
def merkleCalculator(transactions):

def is_valid_transaction(tx): #Transaction Validator function
    if 'vin' not in tx or 'vout' not in tx:
```

```
        return False

    for input_tx in tx['vin']:
        if 'txid' not in input_tx or 'prevout' not in input_tx or 'value' not in input_tx['prevout']:
            return False

    total_input_value = sum(input_tx['prevout']['value'] for input_tx in tx['vin'])
    total_output_value = sum(output_tx['value'] for output_tx in tx['vout'])
    return total_input_value > total_output_value
```

So, this code take transactions from mempool and then validates them.

```
transactions = []
for filename in os.listdir(Mempool_folder):
    if filename.endswith(".json"):
        with open(os.path.join(Mempool_folder, filename), 'r') as f:
            tx = json.load(f)
            if is_valid_transaction(tx):
                transactions.append(os.path.splitext(filename)[0])

print(transactions)                    #to print all txids
print(len(transactions))               #number of validated txids
print(merkleCalculator(transactions))  #will print merkle root!!
#the output will appear as b'63385f87b46b3c6abb56e8041c9cb082c7c94bc919c165cafc8f1311f86399e6'
# where b' denotes byte string in python,we just have to take the string between the commas '____'

version = '01000000' # block version
merkle_root = "63385f87b46b3c6abb56e8041c9cb082c7c94bc919c165cafc8f1311f86399e6"
# calculated via merkleCalculator function
previous_hash = "0000000000000000000000000000000000000000000000000000000000000000"
timestampdec = int(ut)
timestamp = str(hex(timestampdec).lstrip("0x")) #time in hex
difficulty_target = '1f00ffff'# in 4 bytes
target = '0x0000ffff00000000000000000000000000000000000000000000000000000000'
#in 32 bytes

#for nonce
s = version + reverse_byte_order(previous_hash) +
    reverse_byte_order(merkle_root) + reverse_byte_order(timestamp) +
    reverse_byte_order(difficulty_target)
#as merkle root and others were in reverse byte order
#so we again converted then into natural byte order by using reverse_byte_order function
nonce = 0 #initially

while True:
    value = s + str(reverse_byte_order(hex(nonce).lstrip('0x')))
    hash_value = hashlib.sha256(value.encode('utf-8')).hexdigest()
    if int(hash_value, 16)<int(target, 16):      #converting hex into int for comparision
        break
    nonce += 1
```

Now, this code will print the block header in output.txt file in the local folder

---

```
coinbase_tx = {        #Any random hex format
    "txid": SHA256("coinbase"),
    "vin": [{
        "coinbase": "04ffff001d0.................6562756c6c",
        "sequence": 4294967295
    }],
```

```
    "vout": [{
        "value": 5000000000,
        "scriptPubKey": "4104678ad.............2bf11d5fac"
    }]
}

transactions.insert(0, SHA256("coinbase")) # add conibase txid at index 0 !!!!!

block_header = {
    "version": 1,
    "previous_block_hash": previous_hash,
    "merkle_root": merkle_root,
    "timestamp": timestampdec,
    "difficulty_target": "0000"+target.lstrip("0x"), #to remove  0x from hex notation and 0000 is added
    "nonce": nonce
}

with open(OUTPUT_FILE, 'w') as f: # to add all text in output.txt in writng mode
    f.write("Block Header:\n")
    f.write(json.dumps(block_header, indent=2) + "\n\n")
    f.write("Serialized Coinbase Transaction:\n")
    f.write(json.dumps(coinbase_tx, indent=2) + "\n\n")
    f.write("Transaction IDs:\n")
    for i in range(len(transactions)):
        f.write(f"{transactions[i]}\n")

print(nonce) # will print nonce in decimal form
blockHeader = s+str(reverse_byte_order(hex(nonce).lstrip("0x")))
# concatenating nonce as reverse byte order
print(blockHeader+((160-len(blockHeader))*'0'))
#code to print Block Header just to be sure
}
```

You can visit output.txt file

5 Blockchain Technology : Ethereum & Smart Contracts

Ethereum is an open-source blockchain-based platform that enables developers to build and deploy decentralized applications (DApps). Unlike Bitcoin, which primarily serves as digital money, Ethereum focuses on decentralizing other types of applications and services.

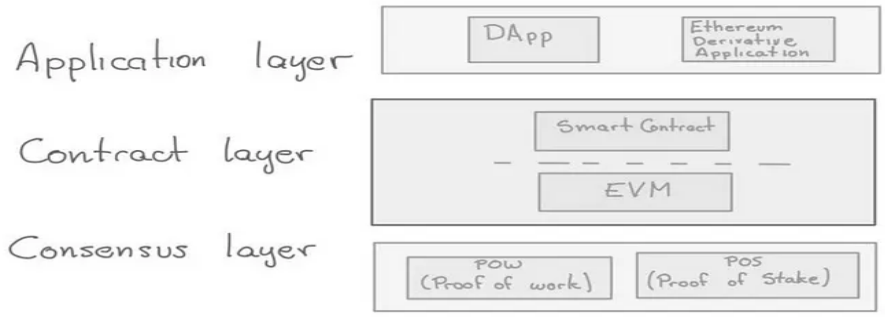
5.1 Ethreum v/s Bitcoin

- **Bitcoin** : -Bitcoin was created as a digital currency to enable peer-to-peer transactions without the need for intermediaries like banks.
  -Primarily used as a store of value and a medium of exchange. Often referred to as "digital gold."
  -It uses PoW mechanism.
- **Ethereum** : -Ethereum was designed as a decentralized platform to facilitate smart contracts and decentralized applications (dApps).
  -Beyond currency transactions (Ether), Ethereum is used for executing complex programmable transactions, creating decentralized applications, and implementing smart contracts.

-It upgraded from the energy-intensive proof-of-work (PoW) to the eco-friendly proof-of-stake (PoS) consensus mechanism after 15th September 2022.

5.2 EVM - Ethereum Virtual Machine

The EVM serves as the decentralized computer that enables the execution of smart contracts and decentralized applications (DApps).Virtual machines allow the same platform to run on many different hardware architectures and operating systems, which makes VMs very suitable for distributed networks like Ethereum.



In addition to being a virtual machine, EVM is also a 'stack machine' and a 'state machine'. The 'state machine' is one that can read inputs and transform them into a new state based on those inputs. The stack-based virtual machine is a virtual machine that organises the memory structure into a stack and accesses it as a stack.

5.3 What is Ether and Gas?

Ether (ETH) is the native cryptocurrency of the Ethereum blockchain. It serves several key purposes within the Ethereum ecosystem :

- Ether can be used as a digital currency for buying goods and services, similar to how Bitcoin is used.
- Like other cryptocurrencies, Ether can be held as an investment.
- Ether is primarily used to pay for computational services on the Ethereum network, which leads to its use in the concept of "gas."

Gas is a fundamental concept in Ethereum, designed to measure the amount of computational effort required to execute operations such as transactions and smart contract functions :

- Gas is used to calculate the fees users must pay to execute transactions and operations on the Ethereum network.
- The gas price is typically specified by the user in Gwei (a small fraction of Ether).Simple operations might cost only a few gas units, while more complex operations cost more.
- Gas is essential to prevent abuse and ensure the efficient allocation of network resources.

5.4 Utility of Ethereum

5.4.1 Smart Contracts

One of the biggest problems with a traditional contract is the need for trusted individuals to follow through with the contract's outcomes but code isn't biased it'll **execute what its logic says as usual** without any trust issues.
Smart contracts are computer programs stored on the blockchain that follow "if this then that" logic, and are guaranteed to execute according to the rules defined by its code, which cannot be changed once created.

5.4.2 Decentralised Apps

DApps, are a type of application that operates on a decentralized network.Smart contracts are the core of DApps. Smart contracts manage the logic and rules of the DApp, ensuring automated and trustless execution.

Benefits of DApps

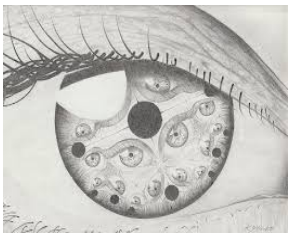
1. Immutability :Once data is recorded on the blockchain, it cannot be altered which provides high level of trust and security.
2. Decentralization : It reduces the risk of censorship or control which is necessary for an app.
3. Transparency : As transactions are recorded on the blockchain which is accessible to every node.
4. Smart Contracts : They can automate the execution of agreements without the need of intermediaries.

5.4.3 Decentralized Finance : DeFi

DeFi allows us to own our money with having to trust third party companies.Bitcoin in many ways was the first DeFi application. DeFi products open up financial services to anyone with an internet connection and they're largely owned and maintained by their users.Not only DeFi we can use web3 to create **DAO**, which is a collectively-owned organization working towards a shared mission without trusting a benevolent leader to manage the funds or operations.

5.4.4 NFTs- Non Fungible Tokens :

Cryptocurrencies (ETH) are the native asset of a specific blockchain protocol, whereas tokens are created by platforms that build on top of those blockchains. NFT is just a way to represent anything unique as an Ethereum-based asset. The uniqueness of each NFT enables tokenization of creative things like art, collectibles real estate or even MEMES.



1489 ETH, Dec 2021

300 ETH, Feb 2021

1697 ETH, June 2021

180 ETH, April 2021

6 Solidity

Solidity is a high-level, object-oriented language, designed for writing and deploying smart contracts on the Ethereum blockchain. Developed specifically for Ethereum, Solidity has become the standard language for creating decentralized applications (DApps) and executing smart contracts.You should have a basic knowledge about OOPS before starting solidity.

6.1 Why Solidity ?

While it's true that many existing programming languages can handle complex logic and computations, Solidity was specifically designed to address the unique requirements and challenges of developing smart contracts on the Ethereum blockchain.

- Gas optimization: Every operation on the Ethereum network consumes gas, which is paid for in Ether (ETH). Solidity includes features that help optimize gas usage, making contracts more efficient and cost-effective.
- Compatibility with the EVM: Solidity is designed to compile into bytecode that operates on the Ethereum Virtual Machine. This close integration ensures that smart contracts can fully utilize the EVM's features
- Built-in functions: Unlike other languages where a logic is needed to be written for every small operation, Solidity provides direct access to blockchain-specific features like sending and receiving Ether, and interacting with other contracts. These functionalities make it very easy to write a Smart Contract.
- Problems with existing languages(they are not designed with the constraints and features of decentralisation,security concerns)

6.2 Basic Layout & Syntax

Term	Description
Bytecode	Bytecode is the low-level code that the Ethereum Virtual Machine (EVM) executes. This bytecode is what gets deployed to the Ethereum network.
Assembly Code	Assembly code in Solidity is a low-level, human-readable form of instructions that closely maps to the EVM bytecode. It helps in debugging and further optimization of code.

SPDX License Identifier : Every source file should start with a comment indicating its license under which the source code is made available.
Pragma: The pragma keyword is used to enable certain compiler features or checks. This means that the compilers of specified version and above are enabled.

Contract : It is similar to class in C++. It contains constructor, state variables, functions and its modifiers. **Inheritance** can also be used to extend functionality of a contract.

Storage : The EVM has three areas where it can store data: storage, memory and the stack.

- **Memory** : Temporary storage used during function execution. Variables declared with memory exist only within the function's scope and are discarded after the function finishes.
- **Storage** : Permanent storage on the blockchain. Variables declared with storage persist even after the function concludes

Data types.: both value(enum,int) and reference(arrays,struct)

- Boolean :accepts True or False only
- int/uint (unsigned) : stores integer values
- address : 20 bytes value to store an Ethereum address
- enum : It can be used to create custom types with a finite set of 'constant values'

```
contract Day {
    enum Time { Morning , Noon, Evening , Night} //enumerator definition
}
```

- struct : unrestricted data type,similar to C

```
contract Election {
    struct Voter { // Structure
        uint voter_id;
        bool voted;
        uint vote;
        address voter_address;
        bool eligibility;
    }
}
```

- array : Arrays: An array is a group of variables of the same data type much like in C++.

```
contract arr {
    uint[5] public fixedArr = [1, 4, 2, 3, 5];
    uint[] public dynamicArr;

    // Function to add a value to the dynamic array
    function addToDynamicArray(uint value) public {
        dynamicArr.push(value);
    }

    // Function to demonstrate the arrays
    function testArray() public view returns (uint[5] memory, uint[]
        memory) {
        return (fixedArr, dynamicArr);
    }
}
```

- string : Stores a bunch of characters.String operation requires more gas as compared to byte operation.

```
contract Literals {
   string data = "test";
   bytes32 data = "test";
}
```

- bytes32 : static strings/stores fixed length of string from range 1 - 32.
- mapping : Mapping stores the data in a key-value pair where a KeyName can be any value type and ValueName can be of any type.
  `mapping(KeyType KeyName => ValueType ValueName) VariableName`
  (KeyName and ValueName are optional.)

```
//use of mapping to update balances
    contract MappingExampleWithNames {
    mapping(address user => uint balance) public balances;

    function update(uint newBalance) public {
        balances[msg.sender] = newBalance;
    }
}
//msg.sender represnets the address of the account that called the
//function.
```

Constructors : A constructor is a special method in any object-oriented language which gets called whenever an object of a class is initialized.
Solidity provides a constructor declaration inside a contract and it invokes only once when the contract is deployed and is used to initialize the contract state. Compiler induces a default constructor when no constructor is explicitly defined.

```
contract SimpleConstructor {
    string public greeting;

    //The constructor takes one argument, _greeting, which is a string stored
        in memory,
    constructor(string memory _greeting){
        greeting = _greeting;
    }
    // Function to retrieve the greeting
    function getGreeting() public view returns (string memory) {
        return greeting;
    }
}
```

Variables : Solidity supports three types of variables.

- State Variables - Variables whose values are permanently stored in a contract storage.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.5.0;

contract Sol_variables {
   // Declaration of state variable
   uint public state_var;

   // constructor
   constructor(){
      state_var = 55;
   }
}
```

OUTPUT : {"0":"uint: 55"} (on Remix IDE)

- Local Variables - Variables whose values are present till function is executing.

```
// SPDX-License-Identifier: GPL-3.0
contract Sol_variables {
    //function definition
   function getResult() public view returns(uint){

      // Initializing local variables
      uint local_1 = 3;
      uint local_2 = 2;
      uint res = local_1 * local_2;

      // Access the local variable
      return res;
   }
}
```

OUTPUT : {"0":"unit: 6"} (on Remix IDE)

- Global Variables - Special variables exists in the global namespace used to get information about the blockchain.
  Examples of these are `block.difficulty (uint), block.gaslimit (uint) , msg.sender, gasleft() returns (uint256)` etc...

Variable Scope : Scope of local variables is limited to function in which they are defined but State variables can have three types of scopes-

- Public - Public state variables can be accessed internally as well as via messages. For a public state variable, an automatic getter function is generated.
- Private - Private state variables can be accessed only internally from the current contract they are defined not in the derived contract from it.
- Internal - Internal state variables can be accessed only internally from the current contract or contract deriving from it without using this.

Control Flow : Its mostly same as in C and C++, go through basic concepts of it.

Functions :

```
function function-name(parameters) visibility type returns(data-types) {
//logic
}
```

Visibility

public - Anyone can call this function.
external - This declares a function that can only be called externally, i.e. from outside the contract.
internal - This declares a function that can only be called from within the contract or its derived contracts.
private - Only this contract can call this function.

Type

view - This function returns data and does not modify the contract's data.A view function can read the state but cannot modify it. Think of it as a function that checks or retrieves the data stored in the contract.

```
contract ViewType {
    // State variable
    uint public storedData;

    // Constructor to set initial value of the state variable
    constructor(uint _initialValue) public {
        storedData = _initialValue;
    }
    // View function to read the state variable
    function getStoredData() public view returns (uint) {
        return storedData;
    }
    // View function to add a number to the state variable
    function add(uint x) public view returns (uint) {
        return storedData + x;
    }
}
```

pure- Function will not modify or even read the contract's data.A pure function does not read from or modify the state. It only performs operations based on its input parameters.

```
contract PureType {
    // Pure function to add two numbers given the input numbers
    function add(uint a, uint b) public pure returns (uint256) {
        return a + b;
    }
}
```

Modifiers : In Solidity, a modifier is always associated with a function. A modifier in programming languages refers to a construct that changes the behavior of the executing code.
`modifier modifierName(datatype argName)` (or it can be without argument)
{
//action to be taken
}

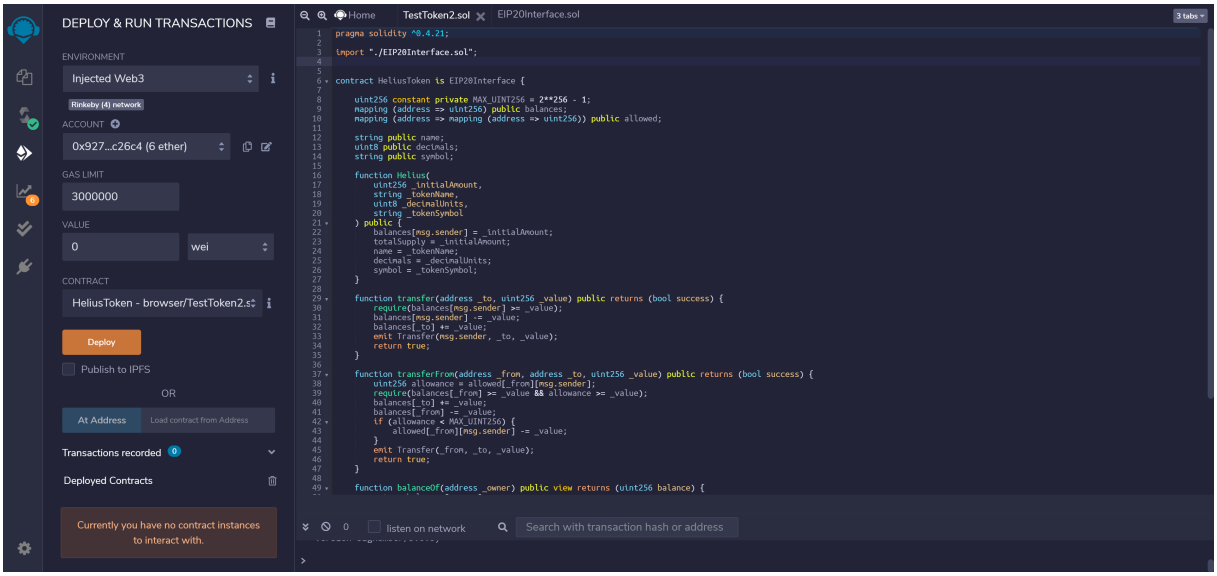
```
    contract AgeRestriction {
    // Modifier to check if the user's age is greater than or equal to the
        specified minimum age for voting
    modifier onlyIfOldEnough(uint256 _age, uint256 _minAge) {
        require(_age >= _minAge, " not old enough");
        _;
    }

    // It can only be executed if the user's age is greater than or equal to
        the specified minimum age
    function restrictedAge(uint256 _age) public pure onlyIfOldEnough(_age, 18)
         returns (string memory) {
        return "You are old enough to vote";
    }
}
//memory: Specifies that the string is a temporary variable, which is cheaper
    in terms of gas and faster to access compared to storage.
```

The _ ; symbol is known as **Merge Wildcard** as after this wildcard the control is moved to the location where the appropriate function definition is located.

6.3 Tools and Environments

6.3.1 Remix IDE

It is a web-based Integrated Development Environment (IDE) for writing, testing, and deploying Solidity contracts and its available for all.



This video is helpful to understand this playground : (from 1:30:00 to 3:30:00)
click here

METAMASK

6.3.2 MetaMask

Metamask is a browser extension that serves as an Ethereum wallet and gateway to DApps and other utilities of web3.
Its is suggested to not use real money here as a beginner. So, we use **faucets**.....(A faucet is website or application which dispenses small amounts of cryptocurrency to the users in order to promote the use of crypto.) **Some faucets may be scams or require users to provide personal information, which can be risky.**
It is suggested to use testnet faucets as they provide fake cryptocurrency and a test blockchain network to experience the chain.
This video is little outdated as the faucet used here is deprecated but you can watch it to get an overview of basic working of wallets: (starts at 26:54)
click here

6.3.3 Hardhat

Hardhat is a development environment designed for Ethereum, offering testing, debugging, and deployment features.

7 Decentralized Bank Smart Contract

Our goal is to develop a decentralized bank smart contract in which the owner can add and transfer money with some functionalities like checking balance.You can test this on Remix IDE as usual.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.7.0 <0.9.0;

contract DBank
{
    address Owner;

    // We are creating the Mapping for the Adding & Transfer Amount in Account
    mapping(address=>uint)Balance;

    // constructor for the address of the Owner
    constructor()
    {
        Owner = msg.sender;
    }

    // function for adding the Ethereum in Account
    function addBalance(uint amount) public returns(uint)
    {
        // first we have to check the is it Owners Account or Not
        require(msg.sender == Owner, "Yes it is Owner Account !!");
        Balance[msg.sender] = Balance[msg.sender] + amount;

        return Balance[msg.sender];
    }

    // function for Get the Balance from an Account
    function getBalance() public view returns(uint)
    {
        return Balance[msg.sender];
    }

    // to transfer the Amount from Owner to Recipient
    function Transfer(address recipient, uint amount) public
    {
        // check the Self account is or not
        require(msg.sender != recipient, "Can't Transfer !! Self Account.");

        // check the owner has balance is available or not
        require(Balance[msg.sender] >= amount, "No, We Can't Transfer.
            Insufficient Balance !!");

        _transfer(msg.sender, recipient, amount);
    }

    function _transfer(address From, address To, uint Amount) private
    {
        Balance[From] = Balance[From] - Amount;
        Balance[To] = Balance[To] + Amount;
    }

}
```

8 Election Smart Contract: Assignment-2

Our goal is to create a robust and secure smart contract for conducting elections on the blockchain. This is one of the best applications of blockchain in real life. (Assignment

Link) This assignment is inspired from the Voting example available on solidity docs : **Click here**
We will use structs, modifiers, functions, events, and access control here.

9 React Framework

React is a popular JavaScript library for building reusable, component-driven user interfaces for web pages or applications.React combines HTML with JavaScript functionality into its own markup language called JSX. React also makes it easy to manage the flow of data throughout the application.
React apps are made out of **components**. A component is a piece of the UI (user interface) that has its own logic and appearance. A component can be as small as a button, or as large as an entire page.
Basic knowledge of HTML,CSS & specifically Javascript is required to grasp React.
Click here

9.1 Basic Syntax

- **JSX Element** : We can define an element and render it on the page by using **ReactDOM.render(componentToRender, targetNode)**, where the first argument is the React element or component that you want to render, and the second argument is the DOM node that you want to render the component to.
  One key difference in JSX is that you can no longer use the word class to define HTML classes. This is because class is a reserved word in JavaScript. Instead, JSX uses className.

```
const JSX_element = (
  <div>
    <h1>Hello React</h1>
    <p> a JSX element</p>
  </div>
);
ReactDOM.render(JSX_element, document.getElementById('id_of_div'))
```

- **React Component** : Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML. There are 2 ways to define components - via class or function.

```
class MyComponent1 extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return(
    <div>
    <h1>Hi, its a class component</h1>
    </div>
    );
  }
};
const MyComponent2 = function() {
  return <h2>Hi, its a function component</h2>;
```

```
}
```

9.2 Hooks

Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

Some popular hooks are given below:

1. **useState** : The React **useState** Hook allows us to track state in a function component.
   State generally refers to data or properties that need to be tracking in an application.

   Syntax: Let this be an example :- const [color, setColor] = useState("");

   The first value, color, is our current state.
   The second value, setColor, is the function that is used to update our state.
2. **useEffect** : The useEffect Hook allows you to perform side effects in your components.
   Some examples of side effects are: fetching data, directly updating the DOM, and timers.

   useEffect accepts two arguments. The second argument is optional.
   useEffect(*function*,dependency)

   That is, the function will render every time the dependency is changed.
3. useCallback : The React useCallback Hook returns a memoized callback function.
   This allows us to isolate resource intensive functions so that they will not automatically run on every render.
   The useCallback Hook only runs when one of its dependencies update.

9.3 Props

Props (short for properties) are a mechanism for passing data from one component to another in React. They enable the creation of dynamic and reusable components.

9.3.1 Using Props

Props are passed to a component similarly to how arguments are passed to a function. When a component is used, props are provided as attributes in the JSX tag. Inside the component, props are accessible through the `props` object.

```
import React from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  return (
    <div>
      <ChildComponent name="John" age={30} />
```

```
      <ChildComponent name="Jane" age={25} />
    </div>
  );
};

export default ParentComponent;
```

Listing 1: Parent Component

```
import React from 'react';

const ChildComponent = (props) => {
  return (
    <div>
      <h1>Hello, my name is {props.name} and I am {props.age} years old.</h1>
    </div>
  );
};

export default ChildComponent;
```

Listing 2: Child Component

In this example, the `ParentComponent` passes two props, `name` and `age`, to the `ChildComponent`. The `ChildComponent` then uses these props to display personalized greetings.

9.3.2 Destructuring Props

To make the code cleaner and more readable, props can be destructured directly in the function parameter list.

```
const ChildComponent = ({ name, age }) => {
  return (
    <div>
      <h1>Hello, my name is {name} and I am {age} years old.</h1>
    </div>
  );
};
```

Listing 3: Destructuring Props

This approach simplifies the component code by eliminating the need to repeatedly reference `props`.

9.3.3 Prop Types

React provides a way to validate the types of props passed to a component through the `prop-types` package. This helps catch bugs and document the intended usage of a component.

```
// Install Prop-Types
npm install prop-types
```

Listing 4: Using Prop-Types

```
import React from 'react';
import PropTypes from 'prop-types';

const ChildComponent = ({ name, age }) => {
```

```
  return (
    <div>
      <h1>Hello, my name is {name} and I am {age} years old.</h1>
    </div>
  );
};

ChildComponent.propTypes = {
  name: PropTypes.string.isRequired,
  age: PropTypes.number.isRequired,
};

export default ChildComponent;
```

In this example, `PropTypes.string` and `PropTypes.number` are used to specify that `name` should be a string and `age` should be a number. The `isRequired` flag indicates that these props must be provided.

9.3.4 Default Props

Default props can be specified to ensure that a component has default values if no props are provided.

```
const ChildComponent = ({ name, age }) => {
  return (
    <div>
      <h1>Hello, my name is {name} and I am {age} years old.</h1>
    </div>
  );
};

ChildComponent.defaultProps = {
  name: 'Anonymous',
  age: 0,
};

export default ChildComponent;
```

Listing 5: Default Props

In this example, if `name` or `age` is not provided, the component will use the default values `'Anonymous'` and `0`, respectively.

9.3.5 Summary

Props are a fundamental concept in React that enable components to be reusable and dynamic. They allow data to flow from parent to child components, help maintain a clean code structure through destructuring, and ensure robustness with prop type validation and default values. Understanding and effectively using props is essential for building scalable and maintainable React applications.

10 React Weather App : Assignment-3

We were told to fetch data from a weather app and display it on the screen, also to implement the login signup functionalities such that only an authenticated user can

search for weather. For frontend you can make your own design or there are many videos available on YouTube.

The github repo link for the same is as follows:ASSIGNMENT SOLUTION LINK