Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Republic of Rwanda
Ministry of Education

RTB | RWANDA TVET BOARD

BLOCKCHAIN FUNDAMENTALS

SWDBF501

Apply Fundamentals of Blockchain

Competence

RQF Level: 5

Learning Hours 100

Credits: 10

Sector: ICT AND MULTIMEDIA

Trade: SOFTWARE DEVELOPMENT

Module Type: Specific Module

Curriculum: CTSWD5003 TVET CERTIFICATE V IN SOFTWARE DEVELOPMENT

Issue Date: February 2024

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Purpose statement	This specific module describes the skills, knowledge and attitude required to Apply Fundamentals of Blockchain. This module is intended to prepare students pursuing TVET Level 5 in software development. Upon completion of this module, the learner will be able **to Design Blockchain system architecture, Apply Solidity Basics, Develop Smart contracts system** and **Apply Frontend Integration**

Elements of competence and Performance Criteria

Elements of competence	Performance criteria
1. **Design blockchain system architecture**	1.1 System requirements are properly identified based on project purpose
	1.2. Blockchain technologies are effectively selected based on project requirements.
	1.3. Architecture of blockchain application is clearly designed based on specific requirements
2. **Apply Solidity**	2.1 Environment is properly prepared based on development tools standards
	2.2 Solidity concepts is clearly applied based on

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Basics	solidity principles
	2.3 Solidity functions are properly created based system requirements
	2.4 Function interaction is accurately implemented based on blockchain technology
	2.5 Gas Costs are accurately optimized according to function definition
3. **Develop Smart contracts system**	3.1 Smart contracts are properly created based on blockchain technology
	3.2. Tokens are properly created based on token standards
	3.3 Security of Smart contracts are properly Applied based on specific requirements
	3.4. Smart contracts are accurately deployed based on specific requirements
	4.1 Web3 dependencies is clearly installed based on

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

4. Apply frontend Integration	versions
	4.2 Smart contract is properly connected based deployed version
	4.3 Functions are clearly used based on smart contract definitions

Course content		
Learning outcome	At the end of the module the learner will be able to: 1. Design blockchain system architecture 2. Apply Solidity Basics. 3. Develop Smart contracts system. 4. Apply frontend Integration	
Learning outcome 1: Design blockchain system architecture	Learning hours: 10 hours	
Indicative content		
Indic. Content 1.1: Identification blockchain requirements I.C.1.1.1 Introduction to blockchain		✓

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- Define:

What is blockchain technology? Blockchain technology is an advanced database mechanism that allows transparent information sharing within a business network. A blockchain database stores data in blocks that are linked together in a chain.

- Blockchain:

Blockchain is a record-keeping technology designed to make it impossible to hack the system or forge the data stored on the blockchain, thereby making it secure and immutable. It's a type of distributed ledger technology (DLT), a digital record-keeping system for recording transactions and related data in multiple places at the same time.

Blockchain technology is a decentralized digital ledger that records transactions in a secure and transparent manner. Each block in the chain contains a unique code and a record of previous transactions.

- Cryptography

Role of Cryptography in Blockchain

The method used to secure data from unauthorised access is called cryptography. Blockchain was built on the concept of enabling secure communication between two parties therefore, it uses cryptography to secure the transactions that take place between two nodes on the network. Blockchain uses two main concepts, cryptography and hashing. The former is used to encrypt messages in a peer-to-peer network, and the latter is used to

secure information on the block and link the blocks in a blockchain.

Maintaining the security of participants and transactions as well as safeguarding against double-spending, is the primary focus of cryptography. It makes sure that the transactions on the blockchain network are secure by ensuring data can be obtained, read and processed only by individuals for whom it was intended.

Definition of Cryptography

Cryptography is the process of developing a set of techniques and protocols to prevent any third party from gaining access to the data during a communication process. Cryptography is a method of securing data from unauthorized access. In the blockchain, cryptography is used to secure transactions taking place between two nodes in a blockchain network. As discussed above, in a blockchain there are two main concepts cryptography and hashing. Cryptography is used to encrypt messages in a P2P network and hashing is used to secure the block information and the link blocks in a blockchain.

The difference between cryptocurrency and blockchain is that cryptocurrency is a digital currency that operates on a blockchain network, which makes it secure and transparent. Or blockchain is usually the underlying technology that enables the creation and management of cryptocurrencies.

. The word 'cryptography' is derived from two Greek terms, 'Kryptos' meaning 'hidden' and 'Graphein' meaning 'to write'.

Some terminologies used in cryptography:

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- **Encryption**

It is the process of conversion of normal text (plaintext) to a random sequence of bits (ciphertext).

- **Decryption**

It is the inverse process of encryption as it involves the conversion of ciphertext to plain text.

- **Cipher**

It is a mathematical function, also known as a cryptographic algorithm, used to convert plaintext to cipher text.

- **Key**

It is the small amount of information required to induce an output from the cryptographic algorithm.

Types of Cryptography

The two types of cryptography are:

- Symmetric-key cryptography.
- Asymmetric-key cryptography.

Let's discuss each of these topics in detail.

1. **Symmetric-key Encryption**: It focuses on a similar key for encryption as well as decryption. Most importantly, the symmetric key encryption method is also applicable to secure website connections or encryption of data. It is also

referred to as secret-key cryptography. The only problem is that the sender and receiver exchange keys in a secure manner. The popular symmetric-key cryptography system is Data Encryption System(DES). The cryptographic algorithm utilizes the key in a cipher to encrypt the data and the data must be accessed. A person entrusted with the secret key can decrypt the data. Examples: AES, DES, etc.

Features:

- It is also known as Secret key cryptography.
- Both parties have the same key to keeping secrets.
- It is suited for bulk encryptions.
- It requires less computational power and faster transfer.

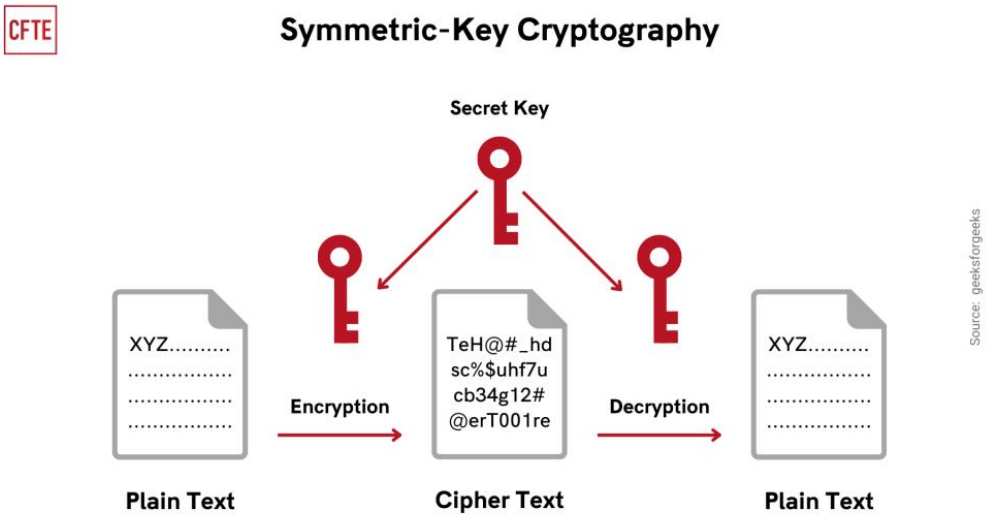
2. **Asymmetric-key Encryption**: This cryptographic method uses different keys for the encryption and decryption process. This encryption method uses public and private key methods. This public key method help completely unknown parties to share information between them like email id. private key helps to decrypt the messages and it also helps in the verification of the digital signature. The mathematical relation between the keys is that the private key cannot be derived from the public key, but the public key can be derived from the private key. Example: ECC,DSS etc.

Features:

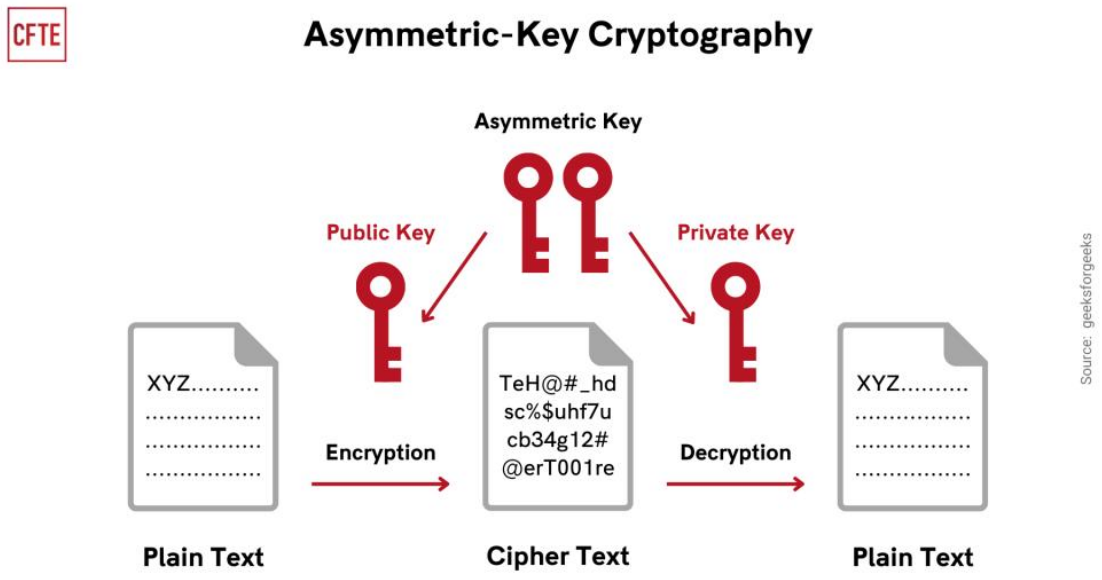
- It is also known as Public-key cryptography.
- It is often used for sharing secret keys of symmetric cryptography.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- It requires a long processing time for execution.
- Plays a significant role in website server authenticity.
- Symmetric-key cryptography



- Asymmetric-key cryptography

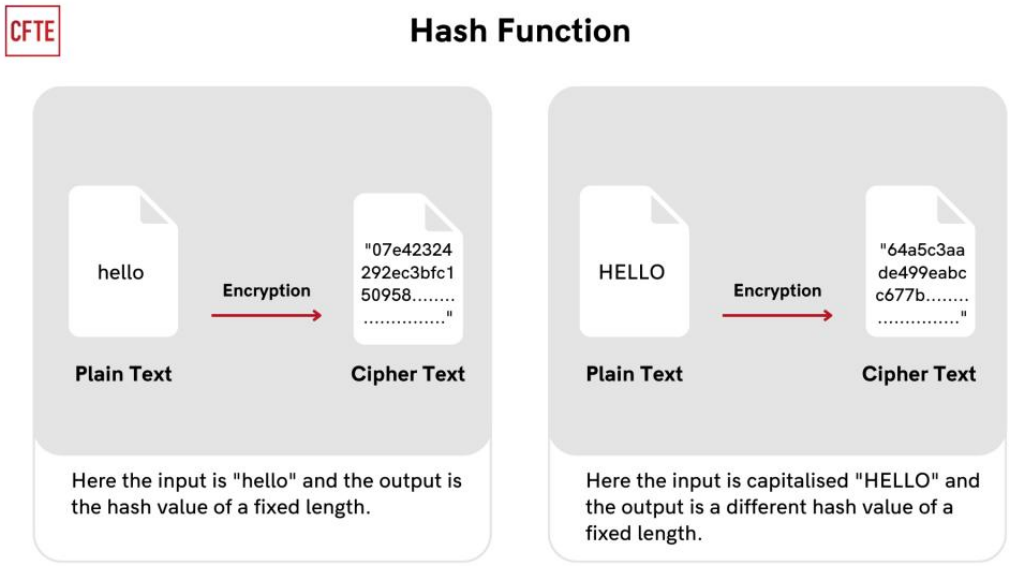


Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Use of Cryptographic Hashing in the Blockchain

Another crucial factor that makes the blockchain secure and immutable is Hashing. Cryptographic hashing is a process that involves coding the data or information on the blockchain into an unreadable, unchangeable and unhackable text.

This method of encryption does not make use of keys but instead uses a cipher to form a hash value of a fixed length from the plaintext. Using a hash algorithm, any plaintext information can be turned into a unique string of text. No matter the length of the input value, the hash always has a fixed length. This process once completed cannot be reversed. Any minor change in the input will result in a different output. This makes threat detection very easy and quick, thus increasing its security.



Benefits of cryptography in blockchain

- ***Reliability***

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

When using cryptographic hashing, transactions are irreversible and thus immutable. This ensures data stored on the blockchain (digital ledger) is reliable, as this process keeps it safe from attackers.

- **Security**

Cryptography is key to keeping the blockchain ledger secure. Every transaction recorded on the blockchain is encrypted, and users can access their data using their private or public keys. Cryptographic hashing makes tampering with data impossible, therefore enhancing its security.

- **Scalability**

The cryptographic hash function allows limitless transactions to be recorded securely on the blockchain network. As several transactions can be combined into one hash, scaling of blockchains is made possible.

Drawbacks (*disadvantages*) of cryptography in blockchain

- **Information is difficult to access**

Heavily encrypted data and digitally signed information make it difficult to access at the most critical times, even for a legitimate user. Any intruder can attack and disable the network.

- **Expensive**

A huge amount of time and money is required for cryptography. Public key encryption involves the setting up and maintenance of its infrastructure for which huge amounts of investment are required. Additionally, adding

cryptographic techniques to send and process information delays the process.

- **Vulnerability**

The level of difficulty and complexity of the mathematical problem used in cryptographic techniques determine the level of security. The less complex the problem, the more vulnerable the cryptographic technique is.

Conclusion

Cryptography is the foundation of blockchain technology. It provides the tools needed to encrypt data, record transactions, and send cryptocurrency securely, all without a centralised authority. It ensures all the blocks get added to the chain without any limit. Hashing in cryptography allows huge amounts of transactions to be stored on the network while protecting them from hackers. Transactions are made safe, reliable and scalable with the help of cryptography.

Role of Cryptography in Blockchain

Blockchain is developed with a range of different cryptography concepts. The development of cryptography technology promotes restrictions for the further development of blockchain.

- In the blockchain, cryptography is mainly used to protect user privacy and transaction information and ensure data consistency.
- The core technologies of cryptography include symmetric encryption and asymmetric encryption.
- Asymmetric cryptography uses digital signatures for verification purposes, every transaction recorded to the block is signed by the sender

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

by digital signature and ensures that the data is not corrupted.

Cryptography plays a key role in keeping the public network secure, so making it fit to maintain the integrity and security of blockchain.

Wallets And Digital Signatures

A blockchain wallet is a special software or a hardware device that is used to keep the transaction information and personal information of the user. Blockchain wallets do not contain the actual currency. The wallets are used to keep private keys and maintain a transaction balance. Wallets are only a communication tool to communicate to carry out transactions with other users. The real data or currency is stored in blocks in the blockchain.

Digital signatures are like proofs that the user gives to the recipient and other nodes in the network to prove that it is a legitimate(*permitted*) node in the network to carry out transactions. While initiating a transaction with other nodes in the blockchain network, the user first has to create a unique digital signature by combining the transaction data with the user's private key using a special algorithm. This process will guarantee the authenticity of the node and the integrity of the data.

Difference Between Symmetric and Asymmetric Encryption

Symmetric and asymmetric encryption are two fundamental cryptographic techniques used to secure data. Here's a comparison between the two:

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Aspect	Symmetric Encryption	Asymmetric Encryption
Definition	Uses a single key for both encryption and decryption.	Uses a pair of keys: a public key for encryption and a private key for decryption.
Key Usage	One key is shared between parties involved.	A public key is shared openly, while a private key is kept secret by the owner.
Speed	Generally faster and requires less computational power.	Slower due to complex mathematical operations.
Security Level	Less secure because the single key must be shared and kept secret.	More secure due to separate keys for encryption and decryption, reducing the risk of key exposure.
Common Algorithms	AES (Advanced Encryption Standard), DES (Data Encryption Standard).	RSA (Rivest-Shamir-Adleman), ECC (Elliptic Curve Cryptography).
Use Cases	Data-at-rest encryption, such as file or database encryption.	Secure communications, digital signatures, and key exchange protocols.
Key Management	Challenging because the key must be securely distributed and kept secret.	Easier key management since only the private key must be kept secure.
Example	Encrypting files on your computer with the same password used for access.	SSL/TLS protocols for secure website connections where a public key encrypts data sent to a server.

Symmetric Encryption: *Fast, simple, but less secure due to key sharing.*

Asymmetric Encryption: *More secure with separate encryption and decryption keys but slower and more computationally intensive.*

Cryptography Hash Function in Blockchain

One of the most notable uses of cryptography is cryptographic hashing. Hashing enables immutability in the blockchain. The encryption in cryptographic hashing does not involve any use of keys. When a transaction is verified hash algorithm adds the hash to the block, and a new unique hash is added to the block from the original transaction. Hashing continues to combine or make new hashes, but the original footprint is still accessible. The

single combined hash is called the root hash. Hash Function helps in linking the block as well as maintaining the integrity of data inside the block and any alteration in the block data leads to a break of the blockchain. Some commonly used hashed function is MD5 and SHA-1.

Properties of Cryptographic Hash:

- For a particular message hash function does not change.
- The input value cannot be guessed from the output hash function.
- They are fast and efficient as they largely rely on bitwise operations.ex. **Cryptography**: Algorithms like RSA and AES make use of bitwise operations for encryption and decryption.

Benefits of Hash function in Blockchain:

1. Reduce the bandwidth of the transaction.
2. Prevent the modification in the data block.
3. Make verification of the transaction easier.

Use of Cryptographic Hash Functions

As the blockchain is also public to everyone it is important to secure data in the blockchain and keeps the data of the user safe from malicious hands. So, this can be achieved easily by cryptography.

- When the transaction is verified through a hash algorithm, it is added to the blockchain, and as the transaction becomes confirmed it is added to the network making a chain of blocks.
- Cryptography uses mathematical codes, it ensures the users to whom

the data is intended can obtain it for reading and processing the transaction.

- Many new tools related to the application of cryptography in blockchain have emerged over the years with diverse functionalities.

Benefits of Cryptography in Blockchain

There are a huge number of benefits of cryptography in blockchain some of them are stated below:

- **Encryption:** Cryptography uses asymmetric encryption to ensure that the transaction on their network guards the information and communication against unauthorized revelation and access to information.
- **Immutability:** This feature of cryptography makes it important for blockchain and makes it possible for blocks to get securely linked by other blocks and also to ensure the reliability of data stored in the blockchain, it also ensures that no attacker can derive a valid signature for unposed queries from previous queries and their corresponding signatures.
- **Security:** Cryptography makes the records of transactions easier using encryption of data, and accessing of data using public and private keys. Cryptographic hashing tampering with data is not possible, making blockchain more secure.
- **Scalability:** Cryptography makes the transaction irreversible(not able to be altered.) giving the assurance that all users can rely on the accuracy of the digital ledger. It allows limitless transactions to be recorded

securely in the network.

- **Non-repudiation(denial):** The digital signature provides the non-repudiation service to guard against any denial of a message passed by the sender. This benefit can be associated with collision resistance i.e.; since every input value has a unique hash function so there is no clash between the messages that are sent and one message can be easily differentiated from the other.
- **Prevent hackers:** The digital signature prevents hackers from altering the data because if the data changes, the digital signature becomes invalid.

Drawbacks of Cryptography in Blockchain

- **Accessibility Issues:** Strong encryption and digital signatures can make data difficult to access in critical situations.
- **Expense:** The cost of implementing and maintaining cryptographic systems can be high.
- **Vulnerability:** The security of cryptographic techniques relies on the complexity of the underlying mathematical problems. Advances in computing can potentially compromise these techniques.

History of blockchain

The history of blockchain is a tale (story) of technological evolution, starting from early cryptographic ideas to its present-day applications across various industries. Here's an overview:

1. Early Foundations (1980s-1990s)

- 1982: David Chaum publishes a paper on blind signatures, a concept for

secure digital payments that later influences blockchain technology.

- 1991: Stuart Haber and W. Scott Stornetta propose a cryptographically secure chain of blocks to timestamp digital documents, ensuring their immutability.

2. Theoretical Concepts and Early Developments (1990s-2008)

- 1998: Wei Dai introduces b-money, an early concept for a distributed digital currency.
- 1999: Hal Finney develops Reusable Proof of Work (RPOW), a system that allows tokens to be exchanged for proof of computational work, laying groundwork for blockchain mining.

3. Bitcoin and the Birth of Blockchain (2008-2012)

- 2008: An individual or group under the pseudonym Satoshi Nakamoto releases the Bitcoin whitepaper, outlining a decentralized digital currency using a blockchain to secure transactions.
- 2009: Nakamoto mines the Genesis Block (Block 0) of Bitcoin, marking the launch of the first blockchain network.
- 2010: The first Bitcoin transaction takes place, and Bitcoin begins gaining attention as a new form of digital currency.

4. Blockchain Expands(become larger) Beyond Cryptocurrency (2013-2017)

- 2013: Vitalik Buterin proposes Ethereum, a platform that introduces smart contracts—self-executing contracts with terms directly written into code.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- 2014: The Ethereum Foundation is established, and Ethereum's ICO (Initial Coin Offering) draws significant investment, leading to the development of a new blockchain platform.
- 2015: The Ethereum blockchain officially launches, expanding blockchain's use beyond cryptocurrency to include decentralized applications (dApps) and smart contracts.

5. Mainstream Adoption and Innovation (2018-2022)

- 2018: The cryptocurrency market experiences a significant boom and subsequent crash, highlighting both the volatility and potential of blockchain technologies.
- 2019: The rise of DeFi (Decentralized Finance) platforms like Uniswap and Aave demonstrates blockchain's capability to offer decentralized financial services.
- 2020: NFTs (Non-Fungible Tokens) gain popularity, showcasing blockchain's potential for unique digital ownership and collectibles.

6. Advancements and Future Trends (2023-Present)

- 2023: Blockchain technology continues to advance with innovations like Layer 2 solutions for scalability, Ethereum 2.0's transition to proof-of-stake for energy efficiency, and increased focus on interoperability between different blockchain networks.
- 2024: Emerging trends include zero-knowledge proofs for enhanced privacy, quantum-resistant algorithms for security, and growing applications in sectors like supply chain management, healthcare, and

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

digital identity.

Types of Blockchain

There are four main types of blockchain:

- **Private blockchain.** Private, or *permissioned*, blockchains require approval to access. These blockchains offer enhanced privacy and control over data, making them suitable for applications that require strict access controls and compliance with regulations. In a private, permissioned blockchain, such as Multichain, every node might be able to perform transactions, but participation in the consensus process is restricted to a limited number of approved nodes.
- **Public blockchain.** Public, or *permissionless*, blockchain doesn't require permission to enter the blockchain network. In a public, permissionless blockchain like Bitcoin, every node in the network can conduct transactions and participate in the consensus process.
- **Hybrid blockchain.** A hybrid blockchain is a type of blockchain that combines the characteristics of permissioned and permissionless blockchains. A hybrid blockchain is set up by a single organization and consists of one public system on top of a private system, giving the organization access control over sensitive data.
- **Consortium blockchain.** Consortium, or federated, blockchain is a type of hybrid blockchain in which a group of organizations governs the blockchain. Consortium blockchains combine the benefits of decentralization and privacy, making them suitable for industries that

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

require collaboration and trust among a select group of participants.

Blockchain Principles

1. Decentralization:

- **No Single Authority:** There's no central authority or intermediary controlling the network.
- **Distributed Network:** Transactions are recorded and verified across a network of computers.
- **Consensus Mechanism:** A consensus mechanism (e.g., Proof of Work, Proof of Stake) ensures that all participants agree on the state of the blockchain.

2. Immutability:

- **Unchangeable Records:** Once a transaction is recorded on the blockchain, it becomes virtually impossible to alter.
- **Blockchain Structure:** The data is stored in blocks, which are linked together in a chain. Altering one block would require changing all subsequent blocks.

3. Transparency:

- **Public Ledger:** The blockchain is a public ledger, meaning anyone can view the transactions.
- **Auditability**: All transactions are transparent and verifiable, providing a high level of transparency and accountability (responsibility).

4. Security:

- **Cryptographic Hash Functions:** Each block contains a cryptographic hash

of the previous block, creating a chain of trust.

- **Private Keys**: Transactions are authorized using private keys, ensuring that only the rightful owner can access and spend funds.

5. Consensus:

- **Agreement on State:** A consensus mechanism ensures that all participants agree on the state (*condition*) of the blockchain.
- **Trustless System:** Participants don't need to trust each other directly, as the consensus mechanism guarantees the integrity(*honesty*) of the system.

Key Applications of Blockchain:

- **Cryptocurrencies:** Bitcoin, Ethereum, and others
- **Supply Chain Management:** Tracking goods from origin to consumer
- **Financial Services**: Smart contracts, decentralized finance (DeFi)
- **Healthcare**: Secure medical records
- **Voting Systems:** Transparent and tamper-proof elections

Functionalities of blockchain

Blockchain technology offers several key functionalities that make it a powerful and versatile*(flexible)* tool. Here are some of the core functionalities:

1. **Decentralization**: Unlike traditional databases that are usually centralized, blockchains distribute data across a network of nodes. This decentralization enhances security and reduces the risk of a single point of failure.
2. **Transparency**: All transactions on a blockchain are recorded in a public ledger that is accessible to all participants. This transparency helps to

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

build trust among users by allowing them to verify transactions independently.

3. **Immutability**: Once data is added to a blockchain, it is extremely difficult to alter or delete. This immutability ensures the integrity and permanence of the transaction history, which is crucial for applications like financial transactions and legal records.
4. **Security**: Blockchains use cryptographic techniques to secure data. Each block contains a cryptographic hash of the previous block, linking them together and making it challenging for malicious actors to alter past data without altering all subsequent blocks, which would require majority control of the network.
5. **Consensus Mechanisms**: Blockchains employ various consensus algorithms (like Proof of Work, Proof of Stake, and others) to agree on the validity of transactions. These mechanisms help ensure that all participants in the network agree on the state of the blockchain without the need for a central authority.
6. **Smart Contracts**: Blockchains can execute self-executing contracts with the terms directly written into code. These smart contracts automatically enforce and execute the terms of an agreement, reducing the need for intermediaries and increasing efficiency.
7. **Tokenization**: Blockchain allows for the creation and management of digital tokens, which can represent assets, currencies, or other value. Tokenization can simplify transactions and enable new business models, such as decentralized finance (DeFi) and non-fungible tokens (NFTs).
8. **Traceability**: Blockchain provides a clear and verifiable record of all transactions. This traceability is beneficial for supply chains, where it helps track the provenance and movement of goods, and for compliance and auditing purposes.
9. **Programmability**: Some blockchains, like Ethereum, support programmable transactions through scripting languages. This

programmability enables the development of decentralized applications (dApps) and complex financial instruments.

What is different Proof of Work and Proof of Stake

Proof of Work (PoW) and Proof of Stake (PoS) are two different consensus mechanisms used by blockchain networks to validate transactions, secure the network, and ensure decentralization without the need for a central authority. Each has a different approach for how participants contribute to the network's security and how new blocks are added to the blockchain.

1. Proof of Work (PoW)

PoW was the first consensus mechanism used in blockchains and is most famously employed by Bitcoin. The primary role of PoW is to secure the blockchain through a competitive, computational process called mining.

2. Proof of Stake (PoS)

Proof of Stake was developed as an energy-efficient alternative to PoW. In PoS, the role of securing the network and validating transactions is based on the amount of cryptocurrency a participant holds and is willing to "stake" or lock up.

Examples of Use:

- Proof of Work: Bitcoin, Litecoin, Ethereum (until its switch to PoS).
- Proof of Stake: Ethereum (post-merge), Cardano, Polkadot, Solana.

Pros and Cons of blockchain

Blockchain technology comes with several advantages and challenges. Here's a breakdown of the key pros and cons:

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Pros (positive aspect)

1. **Decentralization:** Reduces reliance on a central authority.
2. **Transparency:** All participants have access to the same ledger.
3. **Immutability:** Once data is added to the blockchain, it cannot be easily altered or deleted.
4. **Security:** Utilizes cryptographic techniques and consensus mechanisms to secure data, making it highly resistant to hacking and fraud.
5. **Smart Contracts** : Automates and enforces agreements directly through code.
6. **Tokenization**: Facilitates the creation and management of digital assets.
7. **Traceability** : Provides a clear audit trail for assets, which is useful for verifying authenticity and tracking the provenance of goods. Enhances supply chain transparency and accountability.
8. **Data Integrity:** Ensures the accuracy and reliability of data through its immutable and cryptographically secure structure. Reduces risks associated with data tampering and loss.

Cons:/ negative aspects

1. **Scalability:** Many blockchains, especially those using Proof of Work (PoW), face scalability issues. High transaction volumes can lead to slower processing times and increased fees.
2. **Energy Consumption**: Some consensus mechanisms, like PoW, require substantial computational power and energy.
3. **Complexity**: Blockchain technology can be complex and difficult to understand. This complexity can hinder adoption and implementation, especially for those unfamiliar with the technology.
4. **Cost**: Developing and maintaining blockchain solutions can be expensive.
5. **Irreversibility of Mistakes**: Due to the immutable nature of blockchain, mistakes or fraudulent transactions cannot be easily reversed or

corrected once they are recorded.

Blockchain Company solutions

Blockchain companies offer a wide range of solutions to address various business challenges. These solutions leverage the unique features of blockchain technology to provide enhanced security, transparency, efficiency, and trust.

The key areas where blockchain companies are making a significant impact:

1. Supply Chain Management:

- **Traceability:** Blockchain can track products from their origin to the consumer, ensuring transparency and preventing counterfeits.
- **Efficiency:** Smart contracts automate processes, reducing paperwork and streamlining operations.
- **Sustainability:** Blockchain can verify the ethical and sustainable sourcing of materials.

2. Financial Services:

- **Cryptocurrencies:** Blockchain is the foundation of cryptocurrencies like Bitcoin and Ethereum, offering decentralized finance.
- **Smart Contracts:** Smart contracts automate financial transactions, reducing risk and increasing efficiency.

3. Healthcare:

- **Electronic Health Records (EHRs):** Blockchain can secure and share patient data securely and efficiently.
- **Supply Chain:** Blockchain can track the movement of pharmaceuticals, preventing counterfeits and ensuring quality.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

4. Identity Management:

- **Decentralized Identity:** Blockchain can create a secure and portable digital identity, reducing the risk of identity theft.
- **Know Your Customer (KYC):** Blockchain can automate KYC processes, reducing fraud and improving customer onboarding.

5. Voting Systems:

- **Transparency:** Blockchain can ensure the integrity of voting systems, preventing fraud and increasing voter confidence.
- **Accessibility:** Blockchain can make voting more accessible, especially for remote or marginalized populations.

6. Intellectual Property:

- **Ownership Verification:** Blockchain can verify ownership of intellectual property, preventing piracy and counterfeiting.
- **Royalty Management:** Blockchain can automate royalty payments, ensuring fair compensation for creators.

7. Gaming:

- **Non-Fungible Tokens (NFTs):** Blockchain can enable the creation and trading of unique digital assets.
- **In-Game Economies:** Blockchain can create decentralized in-game economies, empowering players and developers.

I.C.1.1.2 Description of blockchain key concepts

- **Essential components of wallet (Private Keys, Public Keys, Addresses)**

In the context of cryptocurrency wallets, understanding the essential

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

components—private keys, public keys, and addresses—is crucial for managing and securing digital assets. Here's a detailed overview of each component:

1. Private Keys

- **Definition:** A private key is a cryptographic key used to sign transactions and provide proof of ownership over a cryptocurrency. It is a critical piece of information that should be kept secure and confidential.
- **Function:** The private key allows the owner to access and manage their cryptocurrency. When a transaction is made, it is signed with the private key to prove that the owner authorizes the transaction.
- **Security:** If someone gains access to your private key, they can control your funds. Therefore, private keys must be stored securely, such as in hardware wallets or encrypted storage.

2. Public Keys

- **Definition:** A public key is derived from the private key using cryptographic algorithms. It is used to generate addresses and can be shared publicly without compromising security.
- **Function:** The public key is used to create a corresponding public address where cryptocurrencies can be received. It also helps in the verification process during transactions, ensuring that the signature matches the public key.
- **Security:** Public keys can be freely shared and are used to receive funds. They do not reveal the private key and thus do not compromise the security of the wallet.

3. Addresses

- **Definition:** An address is a hashed version of the public key. It is used to

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

identify a destination for cryptocurrency transactions.

- **Function:** Cryptocurrency addresses are used to send and receive funds. They are derived from the public key through a series of hashing operations to create a shorter, more manageable identifier.
- **Types:** Addresses vary by cryptocurrency and can come in different formats, such as:
    - **Bitcoin:** Addresses can be in different formats, including Legacy (P2PKH), SegWit (P2SH), and Bech32 (native SegWit).
    - **Ethereum**: Addresses are typically 42 characters long, starting with "0x," derived from the public key.
- **Security:** Addresses themselves do not need to be kept secret and can be shared openly to receive funds. However, they should be used cautiously to avoid sending funds to incorrect or malicious addresses.

How They Work Together:

1. Generating a Wallet:
    - Private Key Generation: A private key is generated randomly.
    - Public Key Derivation: The public key is derived from the private key using cryptographic algorithms.
    - Address Creation: The public key is hashed to create an address.
2. Receiving Funds:
    - Share Address: Provide your public address to the sender.
    - Funds Transfer: The sender uses your address to send funds.
3. Sending Funds:
    - Transaction Creation: Create a transaction and sign it with your private key.
    - Verification: The network verifies the transaction using your public key to ensure it matches the signature.
4. Security Considerations:
    - Private Key Protection: Must be kept confidential and secure to

prevent unauthorized access to your funds.

- Public Key and Address Sharing: Can be shared openly for receiving funds but should be done carefully to avoid errors.

Summary

- **Private Key:** Essential for controlling and accessing your funds; must be kept secret.
- **Public Key:** Derived from the private key; used to create an address and verify transactions.
- **Address:** A user-friendly representation of the public key used to receive funds.

These components work together to ensure secure and efficient management of cryptocurrencies. Understanding and protecting your private key is fundamental to maintaining the security of your digital assets.

Transactions, Merkle Trees, and Blocks

Transactions, Merkle Trees, and Blocks are fundamental concepts in blockchain technology, particularly in the structure of Bitcoin and other cryptocurrencies. Here's an overview of each concept and how they interrelate:

1. Transactions

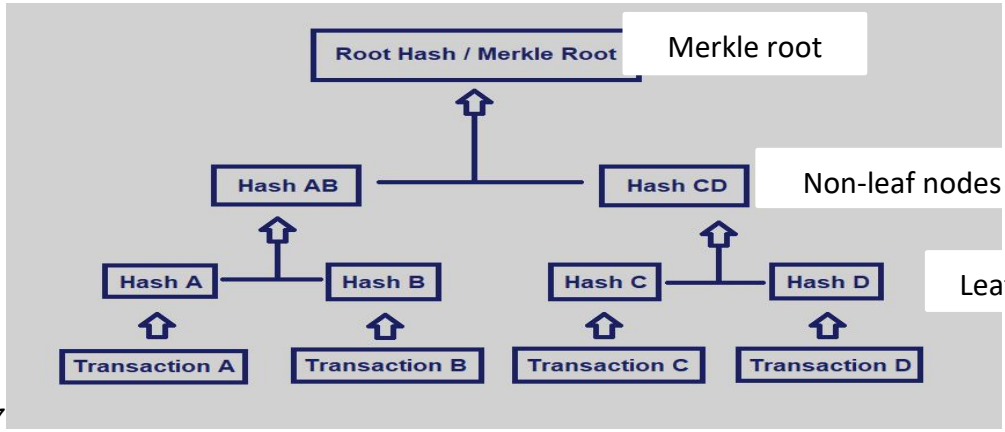
- A transaction in a blockchain is a transfer of value between parties. For example, in Bitcoin, a transaction might represent the transfer of Bitcoin from one user to another.
- Each transaction contains details such as the sender's address, the recipient's address, the amount being transferred, and digital signatures to verify authenticity.
- Transactions are the basic unit of data that are grouped together to

form blocks.

2. Merkle Trees

- A Merkle Tree (or hash tree) is a data structure used to efficiently summarize and verify the integrity of large sets of data.
- In blockchain, Merkle Trees are used to hash and organize transactions within a block. Transactions are hashed, and those hashes are paired and hashed again until a single hash, known as the Merkle Root, is obtained.
- The Merkle Root is a summary of all the transactions in the block and is used to verify the data's integrity without having to download the entire data set. This makes the blockchain more efficient, especially for lightweight clients.

Merkle Tree Structure:



7

- **Leaf nodes:** Represent the hash of individual transactions.
- **Non-leaf nodes:** Represent the hash of the concatenated hashes of their child nodes.
- **Root node (Merkle Root):** The top hash that summarizes all the transactions.

3. Blocks

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- A block is a container that groups multiple transactions. Each block contains a header and a body.
- **Block Header**: Includes metadata such as the previous block's hash (linking blocks together), the Merkle Root (representing all transactions in the block), timestamp, and a nonce used in the mining process.
- **Block Body**: Contains the list of transactions included in the block.
- Blocks are linked together in a chronological order, forming a chain (blockchain), which ensures the security and immutability of the data.

How They Interconnect

- Transactions are grouped into a block.
- Merkle Trees efficiently organize and verify transactions within a block, with the Merkle Root providing a unique fingerprint of the block's transactions.
- The block includes the Merkle Root in its header, which is crucial for validating the integrity of the data.
- Each block links to the previous one through its header, forming the blockchain.

Hierarchical Deterministic Wallets, Mnemonic Seeds and Smart Contracts

A hierarchical deterministic wallet is a crypto wallet that uses a hierarchical structure to derive key pairs (your public and private keys). On a basic level, this allows you to manage multiple accounts with a single crypto wallet.

Advantages of HD Wallets:

- **Simplified Backup**: Only the seed phrase needs to be backed up, making it easier to secure the wallet.

- **Enhanced Security:** The hierarchical structure ensures that compromising one key does not compromise the entire wallet.
- **Improved Privacy:** By generating new addresses for each transaction, HD wallets minimize the chances of linking transactions to a single identity.
- **Organizational Flexibility**: Users can manage multiple accounts, addresses, and transactions in an organized manner.

Mnemonic seeds (or mnemonic phrases) are a way to represent a long string of cryptographic data (such as a private key) in a more human-readable format. They are primarily used in the context of cryptocurrency wallets and other applications involving cryptographic keys. Here's a breakdown of what mnemonic seeds are and how they work:

What is a Mnemonic Seed?

1. A mnemonic seed is a sequence of words (typically 12, 15, 18, 21, or 24 words) that encodes a binary number. This sequence is generated from a random number and can be used to derive a wallet's private keys.
2. Mnemonic seeds are often based on the BIP39 (Bitcoin Improvement Proposal 39) standard, which specifies how to create and use mnemonic phrases in cryptocurrency wallets.

Benefits of Mnemonic Seeds

- **Ease of Use:** Instead of remembering a long string of numbers and letters, users can remember a simple phrase of words.
- **Backup and Recovery:** If a user loses access to their wallet, they can recover it using the mnemonic phrase.
- **Portability:** Mnemonic seeds can be easily written down and stored in a

safe place.

Security Considerations

- **Keep It Safe:** Anyone with access to your mnemonic seed can access your cryptocurrency wallet and funds. It's important to store it securely, offline if possible.
- **Do Not Share:** Never share your mnemonic phrase with anyone. Scammers may try to trick you into giving it away.

Smart contracts are self-executing agreements with the terms of the contract directly written into code. They run on blockchain platforms, most notably Ethereum, and automatically enforce the contract terms when predefined conditions are met. Smart contracts revolutionize how agreements are made and executed by eliminating intermediaries, reducing costs, and increasing efficiency and transparency.

How Smart Contracts Work:

- Smart contracts are coded using programming languages such as Solidity (for Ethereum). The code defines the rules and penalties of the agreement, along with automated actions triggered by specific events.
- Once deployed on a blockchain, smart contracts are immutable and publicly accessible. They execute automatically when the required conditions are fulfilled, such as transferring funds or verifying information.
- The decentralized nature of blockchains means that no central authority is needed to verify the execution of the contract. Instead, the contract operates in a trustless manner, relying on the consensus of the network.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Platforms Supporting Smart Contracts:

- **Ethereum:** The most widely used blockchain for smart contracts, known for its robust ecosystem and developer tools.
- **Binance Smart Chain (BSC):** A popular alternative to Ethereum with lower transaction fees and faster processing times.
- **Cardano, Solana, Polkadot:** Other emerging platforms offering scalability and unique approaches to smart contract functionality.

Working of Blockchain Transaction

Blockchain transactions are the process by which data is added to a blockchain ledger in a secure, transparent, and immutable way.

1. Sender initiates a transaction.
2. Transaction is broadcasted to the network.
3. Nodes verify the transaction.
4. Valid transactions go to the mempool.
5. Miners create a new block with transactions.
6. The block is added to the blockchain.
7. Other nodes verify and confirm the block.
8. Receiver sees the updated balance.
9. Transaction is recorded immutably on the blockchain.

Explanation show transaction between Alice And Bob

Step 1: Transaction Initiation

- Sender: The process begins when the sender initiates a transaction. For example, Alice wants to send 1 Bitcoin to Bob.
- Details: Alice specifies Bob's wallet address and the amount.

Step 2: Broadcasting the Transaction

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- Network Broadcast: Alice's transaction is broadcasted to the blockchain network, which consists of multiple nodes (computers).

Step 3: Transaction Verification

- Validation: Nodes in the network validate the transaction. They check:
  - If Alice has enough balance.
  - If the transaction adheres to the blockchain's rules.

Step 4: Transaction Pool

- Mempool: Validated transactions are placed in a pool (mempool) where they await confirmation.

Step 5: Block Creation

- Mining: Miners (nodes that validate and add transactions to the blockchain) select transactions from the mempool to include in a new block. Miners compete to solve complex mathematical problems to add the block.
- Proof of Work: The first miner to solve the problem gets to add the block and is rewarded (e.g., with Bitcoin).

Step 6: Block Addition to the Blockchain

- Linking Blocks: Once a miner successfully creates a block, it is added to the existing blockchain. Each block contains a cryptographic hash of the previous block, linking them together securely.

Step 7: Confirmation

- Network Consensus: Other nodes in the network verify the new block and reach consensus. Once confirmed, the transactions within the block

are considered finalized.

- Number of Confirmations: Often, multiple confirmations (additional blocks added) are required for higher security.

Step 8: Transaction Completion

- Receiver's Wallet: Bob's wallet is updated to reflect the received Bitcoin. He can now see the transaction in his wallet.

Step 9: Record Keeping

- Immutable Ledger: The transaction is recorded on the blockchain, creating a permanent and tamper-proof record that can be viewed by anyone.

Example: Life Cycle of a Bitcoin Transaction

1. Transaction Creation: Alice wants to send 0.5 BTC to Bob. She creates a transaction, signs it with her private key, and broadcasts it to the Bitcoin network.
2. Transaction Propagation: The transaction propagates across the Bitcoin nodes. Each node validates the transaction by checking Alice's signature and her account balance.
3. Mempool: The transaction enters the mempool, waiting for a miner to pick it up.
4. Mining: Miners compete to solve a complex cryptographic puzzle. The winning miner includes Alice's transaction in the next block.
5. Consensus: The block is validated by the majority of nodes, reaching consensus. The block is added to the Bitcoin blockchain.
6. Confirmation: Alice's transaction receives its first confirmation when the block is added. After six confirmations (six new blocks), Bob can be

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

confident the transaction is secure.

7. Transaction Finalization: The transaction is irreversible, and Bob's Bitcoin wallet balance is updated with the 0.5 BTC.

Key Components in the Transaction Life Cycle

1. Digital Signature and Encryption: Ensures that only the legitimate owner of the private key can sign and send transactions, providing security.
2. Mempool: Temporary storage for unconfirmed transactions.
3. Block: A container that holds a group of validated transactions.
4. Consensus Algorithm: The mechanism used to agree on the validity of transactions and blocks (e.g., Proof of Work, Proof of Stake).
5. Mining/Validation: The process by which new blocks are created and transactions are confirmed.
6. Confirmations: The number of subsequent blocks added after the transaction's block, providing increased security and finality.

Apply blockchain use cases

Blockchain technology has a wide range of applications across various industries due to its decentralized, transparent, and immutable nature. Here are some key use cases where blockchain is applied:

1. Cryptocurrencies:

- Bitcoin and Altcoins: The most well-known use case of blockchain is cryptocurrencies like Bitcoin, Ethereum, and various altcoins. Blockchain provides a secure, decentralized ledger for recording transactions and managing digital assets without the need for a central authority.

2. Supply Chain Management:

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- Traceability: Blockchain helps track the origin, journey, and handling of goods from production to delivery. This enhances transparency, reduces fraud, and ensures product authenticity.
- Verification: Companies can verify the integrity of their supply chains and ensure compliance with regulatory standards.

3. Financial Services:

- Cross-Border Payments: Blockchain simplifies and speeds up cross-border transactions by removing intermediaries and reducing costs associated with traditional banking systems.
- Smart Contracts: Automate and enforce contract terms for financial agreements, such as insurance policies, derivatives, and loans, without intermediaries.

4. Healthcare:

- Patient Records: Blockchain provides a secure and interoperable system for managing patient records, ensuring privacy, and enabling access to medical history across different healthcare providers.
- Drug Traceability: Tracks the manufacturing and distribution of pharmaceuticals to prevent counterfeiting and ensure the authenticity of drugs.

5. Voting Systems:

- Secure Voting: Blockchain can be used to create secure and transparent voting systems, reducing the risk of fraud and ensuring that votes are accurately counted and recorded.
- Election Integrity: Enhances trust in election processes by providing a tamper-proof record of votes.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

6. Identity Management:

- *Digital Identities: Blockchain can be used to create decentralized digital identities that users control, providing a secure way to verify and manage personal information.*

7. Energy Sector:

- *Energy Trading: Blockchain enables peer-to-peer energy trading, allowing individuals and businesses to buy and sell energy directly, often using renewable sources.*

8. Legal Sector:

- *Smart Contracts: Automate and enforce legal agreements, reducing the need for traditional legal processes and intermediaries.*
- *Evidence Management: Provides a tamper-proof record of legal documents and evidence, ensuring their integrity and authenticity.*

Benefits of Blockchain Use Cases:

- *Decentralization: Reduces the need for central authorities and intermediaries, enhancing efficiency and reducing costs.*
- *Transparency: Provides a transparent and immutable record of transactions and activities, improving trust and accountability.*
- *Security: Uses cryptographic techniques to secure data and protect against tampering and fraud.*
- *Efficiency: Streamlines processes and reduces delays by automating tasks and eliminating intermediaries.*

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

IC 1.2: Selecting Blockchain Technologies

1.2.1:Description of Blockchain technology stack principles

- **Consensus Layer (PoW,PoS, etc)**

The Consensus Layer is a critical component of blockchain technology that

ensures agreement on the state of the blockchain among distributed nodes. It establishes the rules and processes for validating and confirming transactions, adding new blocks to the blockchain, and maintaining the network's integrity.

Different consensus mechanisms achieve this in various ways. Here's an overview of the most common consensus mechanisms:

1. Proof of Work (PoW):

- Overview: PoW is the original consensus mechanism used by Bitcoin and many other blockchains. It requires participants (miners) to solve complex mathematical puzzles(***solutions***) to validate transactions and create new blocks.
- Process:
    - Miners compete to solve a cryptographic problem, known as a hash puzzle, which requires significant computational power.
    - The first miner to solve the puzzle broadcasts the solution to the network.
    - Other nodes verify the solution and, if valid, the new block is added to the blockchain.

2. Proof of Stake (PoS):

- Overview: PoS is an alternative to PoW that selects validators based on their stake in the network rather than computational power. Validators are chosen to create new blocks and confirm transactions based on the amount of cryptocurrency they hold and are willing to "stake" as collateral.
- Process:
    - Validators are selected to propose and validate new blocks based on their stake and other factors like age of the stake or random

selection.
    - Validators receive transaction fees or rewards for their work and are penalized if they act maliciously.

3. Delegated Proof of Stake (DPoS):

- Overview: DPoS is an evolution of PoS that introduces a layer of delegation. Token holders elect a small number of delegates or representatives who are responsible for validating transactions and creating new blocks.
- Process:
    - Token holders vote for delegates who will perform the validation tasks.
    - Delegates take turns producing blocks and validating transactions according to the voting power they hold.

4. Proof of Authority (PoA):

- Overview: PoA is a consensus mechanism where a limited number of trusted nodes, known as authorities, are responsible for validating transactions and creating new blocks. It is often used in private or consortium blockchains.
- Process:
    - Authority nodes are pre-approved and have their identities verified.
    - These nodes take turns creating blocks and validating transactions.

5. Proof of Space and Time (PoST):

- Overview: PoST combines the concepts of space and time, where participants prove that they have allocated disk space and elapsed time

to participate in the consensus process.

- Process:
  - Participants allocate disk space to store cryptographic proofs.
  - Proofs are periodically verified over time to confirm participation and contribute to block validation.

6. Proof of Elapsed Time (PoET):

- Overview: PoET is used in some blockchain networks to achieve consensus by ensuring that each participant waits for a random period before proposing a new block. It is often used in permissioned blockchains.
- Process:
  - Participants are randomly selected to wait for a period before they are allowed to propose a new block.
  - The selection process is managed by a trusted execution environment (TEE) to ensure fairness.

Network Layer (Ethereum's Peer-to-Peer Network)

In Ethereum, the Peer-to-Peer (P2P) network layer is responsible for connecting nodes, ensuring they share and propagate data (transactions, blocks, etc.), and maintaining the integrity and consistency of the blockchain.

Node Communication: Nodes use the Ethereum P2P protocol (DevP2P) to connect and exchange data over the internet. This protocol ensures secure and efficient communication between nodes.

Discovery and Peering: Ethereum nodes discover and connect with each other using the Discovery Protocol (Kademlia DHT). Nodes select peers based on various factors to optimize data sharing.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Data Propagation: When a transaction or block is created, it is broadcast to the network. Nodes validate and propagate this data to ensure it reaches all participants quickly.

Synchronization: Nodes synchronize with the blockchain and the Ethereum Virtual Machine (EVM) to stay updated with the latest state and transactions.

Message Types: Includes transactions, blocks, status updates, and control messages for managing network connections and operations.

Protocols and Libraries: Utilizes protocols like DevP2P for communication and RLPx for data transmission, with Discovery v4 for peer discovery.

Challenges: Addresses scalability, latency, and security through various techniques, including sharding and layer 2 solutions.

- **Protocol Layer (Ethereum's EVM -Ethereum Virtual Machine)**

The Ethereum Virtual Machine (EVM) is a software runtime environment that executes smart contracts on the Ethereum blockchain. It's a crucial component of the protocol layer, responsible for interpreting and executing the bytecode of smart contracts.

Key Functions of the EVM:

- **Smart Contract Execution:** The EVM processes the bytecode of smart contracts and executes their instructions.
- **State Management:** It maintains the state of the Ethereum blockchain, including the balances of accounts and the storage of smart contracts.
- **Transaction Processing:** The EVM validates and processes transactions, ensuring that they comply with the rules of the Ethereum network.
- **Gas Consumption:** It measures the computational resources used by smart contracts and charges gas fees to incentivize miners to include

them in blocks.

How Does the EVM Work?

1. **Bytecode Compilation:** Solidity or other high-level programming languages are used to write smart contracts. These contracts are then compiled into bytecode, which is a sequence of machine-level instructions.
2. **Transaction Submission:** A transaction containing the bytecode of a smart contract is submitted to the Ethereum network.
3. **EVM Execution:** The EVM receives the transaction and executes the bytecode within a sandboxed environment. This prevents smart contracts from interfering with the core functionality of the Ethereum network.
4. **State Updates:** If the smart contract execution is successful, the EVM updates the state of the blockchain accordingly.
5. **Gas Fee Calculation:** The EVM calculates the gas consumed by the smart contract and charges the sender a fee based on the gas price and gas limit set in the transaction.

Importance of the EVM:

- **Decentralized Applications (DApps):** The EVM enables the creation of decentralized applications that can be built on top of the Ethereum blockchain.
- **Smart Contract Functionality:** It provides the foundation for the execution of smart contracts, which can automate various processes and agreements.
- **Flexibility and Extensibility:** The EVM's architecture allows for the development of new programming languages and tools for creating smart contracts.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Smart Contracts Layer (Decentralized Finance (DeFi) Platforms)

The **Smart Contracts Layer** in Decentralized Finance (DeFi) platforms refers to self-executing contracts with the terms of the agreement written directly into code. These contracts automatically enforce and execute financial transactions like lending, borrowing, trading, or staking, without the need for intermediaries like banks.

Here's how it works:

- **Automation**: Once conditions coded in the smart contract are met (e.g., someone deposits funds), the contract executes the corresponding action (e.g., transferring those funds, paying interest).
- **Transparency**: The rules and transactions are visible on the blockchain, making it transparent and auditable.
- **Security**: Once deployed, smart contracts can't be easily changed, reducing the risk of fraud.

DeFi platforms use these smart contracts to offer financial services that are decentralized, more accessible, and often lower cost.

Application Layer (CryptoKitties)

The Application Layer is where users interact with decentralized apps (dApps) that are built on top of blockchain systems. A well-known example of a dApp is CryptoKitties.

What is CryptoKitties?

CryptoKitties is an online game where you can buy, collect, breed, and trade virtual cats. Each cat is unique and is stored as a non-fungible token (NFT) on

the Ethereum blockchain, meaning no two CryptoKitties are the same.

Here's how it works in simple terms:

- Buying and Selling: You can trade your CryptoKitties with other players in a marketplace. Since the game is built on blockchain, the ownership and transactions are recorded securely.

Why It's Important:

- Unique Ownership: Because each CryptoKitty is an NFT, you truly own it. Even the game creators can't take it away from you.
- Decentralized System: The game runs on Ethereum, meaning no single company controls everything. Transactions (like buying, selling, or breeding cats) are handled automatically through smart contracts.

Storage Layer (IPFS – InterPlanetary File System)

The Storage Layer is where large files and data are stored in a decentralized system. The InterPlanetary File System (IPFS) is a tool used to store and share files in a way that doesn't rely on one main server. Instead, files are split into pieces and stored across many computers around the world.

Here's how it works in simple terms:

- Decentralized Storage: Files are not kept in one place. Instead, they are stored on different computers (nodes) all over the world. This makes it hard for the data to be lost or taken down.
- Unique File Addresses: Every file is given a unique code, called a hash, which acts like a fingerprint for the file. When you want to access a file,

you use this code, and the system finds the file for you.

- Efficient Storage: If many people upload the same file, IPFS only stores it once to save space.
- Blockchain Connection: In blockchain systems (like Ethereum), the actual file is stored on IPFS, and only the unique file code (hash) is saved on the blockchain. This way, the blockchain doesn't get overloaded with large files.

Common Uses of IPFS:

- NFTs: Digital art and collectibles use IPFS to store the artwork, while the blockchain keeps the ownership details.
- Decentralized Websites: Websites can be hosted on IPFS, making them harder to block or shut down.
- File Sharing: People can share files directly using IPFS without needing a central platform like Dropbox.

Identity and Access Management (SelfKey)

SelfKey allows users to take control of their digital identities through a decentralized system, offering secure, private, and efficient identity verification processes for accessing various services.

How SelfKey Works:

1. Download the SelfKey Identity Wallet: Users download the wallet and store their identity documents securely.
2. Identity Verification: Users can verify their identity through accredited providers or submit documents to relevant services.
3. Use Identity for Services: The verified identity can then be used to interact with various services on the SelfKey Marketplace, such as

banking, financial services, legal incorporation, or obtaining residency.

- **Security and Encryption (Public and Private Key Encryption)**

  Public and private key encryption is a cryptographic system that uses a pair of keys to secure and authenticate digital communications. The public key encrypts data, while the private key decrypts it, ensuring secure, private, and authenticated exchanges of information.

- **Interoperability Layer (Polkadot)**

  Polkadot's interoperability layer allows different blockchains (parachains) to communicate and share assets securely and efficiently. Its relay chain and bridge system make it possible for both Polkadot-based and external blockchains to interact, enhancing scalability, security, and functionality across the blockchain ecosystem.

- **Scalability Solutions (Lightning Network for Bitcoin)**

The Lightning Network is a scalability solution for Bitcoin designed to enable faster and cheaper transactions by moving small payments off-chain while still leveraging the security of the Bitcoin blockchain. It addresses Bitcoin's limitations in handling a large volume of transactions, making microtransactions more efficient.

Key Features of the Lightning Network:

1. Off-Chain Transactions:
   - The Lightning Network allows users to transact off-chain (outside the Bitcoin blockchain), significantly reducing the number of transactions that need to be recorded on-chain.
   - Only the opening and closing of payment channels are recorded on

the Bitcoin blockchain, while intermediate transactions happen off-chain.

2. Payment Channels:
    - Users open a payment channel by creating a multi-signature Bitcoin transaction, which is recorded on the Bitcoin blockchain.
    - Once a channel is open, they can send and receive an unlimited number of transactions between them without needing further blockchain confirmations.
    - When the channel is closed, the final balance is settled on-chain.
3. Micropayments:
    - The Lightning Network is ideal for small, frequent payments, such as buying digital content, tipping, or paying for services where low transaction fees are essential.
    - It reduces the need to pay the high fees associated with on-chain Bitcoin transactions, especially when the network is congested.
4. Instant Transactions:
    - Payments are nearly instantaneous on the Lightning Network because they do not require the 10-minute confirmation time needed for Bitcoin on-chain transactions.
    - This makes Bitcoin more suitable for everyday purchases, like coffee or retail shopping, where waiting for transaction confirmations is impractical.
5. Low Fees:
    - By settling transactions off-chain, the Lightning Network significantly lowers transaction fees, making microtransactions economically viable.
    - Fees are only incurred when channels are opened or closed on the Bitcoin blockchain, with most transactions inside the payment channels being essentially free or very low-cost.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

6. Scalability:
    - The Lightning Network dramatically increases Bitcoin's transaction throughput. Since most transactions occur off-chain, the Bitcoin blockchain can focus on larger and less frequent transactions.
    - It helps Bitcoin scale to support millions of transactions per second, compared to the limited capacity of on-chain transactions.
7. Routing:
    - If two users do not have a direct payment channel, the Lightning Network can route transactions through multiple channels until it reaches the intended recipient.
    - This enables payments to move across the network without the need to open a direct channel with every participant.

How it Works:

1. Opening a Channel: Two users create a multi-signature Bitcoin address (a payment channel) and lock a certain amount of Bitcoin into it.
2. Off-Chain Transactions: Inside this payment channel, they can make numerous microtransactions by updating their individual balances off-chain.
3. Closing the Channel: When they decide to close the channel, the final state of the balances is broadcast to the Bitcoin blockchain, and the funds are settled accordingly.

Use Cases:

- Micropayments: Lightning Network is ideal for small, everyday transactions, such as paying for coffee, tipping online creators, or small online purchases.
- Cross-Border Payments: Lightning enables fast, low-cost cross-border payments without relying on traditional banking systems.

- Merchant Payments: Businesses can accept Bitcoin payments quickly and cheaply without waiting for blockchain confirmations.

Governance Mechanisms (Tezos)

Tezos is a blockchain platform known for its innovative on-chain **governance mechanisms**, which allow the network to upgrade and evolve without requiring hard forks. This approach ensures that Tezos remains adaptable, secure, and community-driven while minimizing disruptions.

Use Cases of Governance:

- **Protocol Upgrades:** Tezos has successfully upgraded multiple times since its launch, introducing improvements such as lower gas fees, enhanced smart contract functionality, and improved consensus algorithms.
- **Decentralized Decision-Making**: Token holders can vote on any aspect of the protocol, from technical changes to governance rules themselves, making the system adaptable and reflective of the community's desires.

User Interfaces (MetaMask)

MetaMask is a popular cryptocurrency wallet and gateway to decentralized applications (dApps) on the Ethereum blockchain.

Its user interface (UI) is designed to be simple and accessible for both beginners and advanced users, enabling easy interaction with blockchain technology.

Key Features of MetaMask's User Interface:

1. Browser Extension & Mobile App:
    - MetaMask is available as a browser extension (for Chrome, Firefox, Brave, etc.) and as a mobile app (iOS and Android). This allows users to manage their cryptocurrency and interact with dApps

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

directly from their browser or mobile device.

2. Simple Account Management:
    - The UI offers easy setup for creating or importing an Ethereum wallet. Users can create new wallets, back them up with a seed phrase, and manage multiple Ethereum accounts within one interface.
    - Switching between accounts is straightforward, making it convenient for users to manage different wallets or identities.
3. Send and Receive Tokens:
    - The wallet interface allows users to easily send and receive Ethereum or ERC-20 tokens.
    - The "Send" function includes fields for inputting the recipient's address, the amount, and gas fees, which can be adjusted based on the user's preference for transaction speed.
4. Token Management:
    - MetaMask automatically detects commonly used tokens, and users can manually add custom tokens by entering their contract address, making it flexible for managing a wide range of tokens.
5. Connection to Decentralized Applications (dApps):
    - MetaMask allows users to seamlessly connect with dApps through its in-browser extension or mobile app.
    - When visiting a dApp, users can approve or deny requests to connect their MetaMask wallet. This integration lets users perform actions like token swaps, NFT purchases, or participating in decentralized finance (DeFi) without leaving the browser or app.
6. Transaction History:
    - Users can view a detailed history of their transactions, including pending, successful, and failed transactions.
    - This transparency allows users to track their activity and

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

troubleshoot any issues.

7. Gas Fee Management:
    - MetaMask offers the ability to adjust gas fees before confirming transactions, giving users control over the transaction speed and cost.
    - The UI provides basic (slow, average, fast) or advanced options for customizing gas limits and fees.
8. Network Switching:
    - MetaMask supports different Ethereum-based networks, including mainnet, testnets (like Ropsten and Kovan), and custom RPC networks (such as Binance Smart Chain or Polygon).
    - Users can easily switch between these networks via the UI, enabling access to a wide variety of blockchains and dApps.
9. Security & Privacy Controls:
    - The wallet offers security features like password protection, seed phrase backup, and hardware wallet integration (e.g., with Ledger or Trezor).
    - MetaMask allows users to maintain privacy by not sharing account information with dApps unless explicitly authorized.

1.2.2:Describe types of Consensus mechanism

- Proof of Work
- Proof of Stake
- Delegated Proof of Stake
- Proof of Authority
- Proof of Weight

Consensus mechanisms are protocols used in blockchain networks to achieve

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

agreement on the state of the blockchain among distributed nodes. They ensure that all participants in the network validate and agree on transactions, maintaining the integrity and security of the blockchain.

Here are some common types of consensus mechanisms:

1. Proof of Work (PoW):

- Overview: Miners compete to solve complex mathematical problems to validate transactions and create new blocks.
- How It Works: The first miner to solve the problem gets to add the block to the blockchain and receives a reward in cryptocurrency.
- Examples: Bitcoin, Ethereum (prior to Ethereum 2.0).
- Pros: High level of security and decentralization.
- Cons: Energy-intensive, slower transaction speeds, and potential for centralization due to mining pools.

2. Proof of Stake (PoS):

- Overview: Validators are chosen to create new blocks based on the number of coins they hold and are willing to "stake" as collateral.
- How It Works: The more coins a validator stakes, the higher the chance they have of being selected to validate transactions and create a new block.
- Examples: Ethereum 2.0, Cardano, Tezos.
- Pros: More energy-efficient than PoW, faster transaction speeds, and lower barriers to entry.
- Cons: Risk of centralization if wealth concentration occurs, potential for "nothing at stake" issues.

3. Delegated Proof of Stake (DPoS):

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- Overview: Stakeholders elect a small number of delegates or witnesses who validate transactions on their behalf.
- How It Works: Users vote for delegates based on their stake in the network. The elected delegates take turns producing blocks.
- Examples: EOS, Tron, BitShares.
- Pros: Faster transaction times and higher scalability.
- Cons: Centralization risk due to a limited number of delegates and potential for collusion.

4. Proof of Authority (PoA):

- Overview: A consensus mechanism where a limited number of nodes, known as validators, are authorized to create new blocks based on their identity.
- How It Works: Validators are pre-selected and trusted, and they validate transactions based on their reputation.
- Examples: VeChain, PoA Network.
- Pros: Fast transaction speeds and lower energy consumption.
- Cons: Centralization risk due to reliance on trusted validators and less decentralized.

Proof of Weight is a consensus mechanism designed to enhance blockchain scalability and security by introducing a new way to determine the probability of a node being chosen to create the next block, based on a combination of several factors.

- ✓ Identify the types of attacks and vulnerabilities of blockchain

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- Attack in consensus mechanism
- Sybil Attack (spamming the network, disrupt communication among nodes)
- Double Spending
- Eclipse Attack
- Smart Contract Vulnerabilities (re-entrancy attacks, integer overflow/underflow).
- DDoS Attack (Distributed Denial of Service)
- Blockchain Spamming
- Selfish Mining
- Routing Attacks
- Consensus Manipulation

1. Attack in Consensus Mechanism

"attack" in a consensus mechanism refers to any malicious action aimed at disrupting the consensus process. Consensus mechanisms are protocols that ensure all participants in a network agree on the state of the blockchain or distributed ledger.

2. Sybil Attack:

An attacker creates multiple identities (nodes) to manipulate the network's consensus process.

3. Double Spending

This vulnerability occurs when a user attempts to spend the same cryptocurrency more than once.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

4. Eclipse Attack

An eclipse attack is a type of cyberattack primarily associated with distributed systems and blockchain networks.

5. Smart Contract Vulnerabilities

Smart contracts are self-executing contracts with the terms directly written into code. Vulnerabilities can lead to exploitation.

Smart contract vulnerabilities refer to weaknesses or flaws in the code of smart contracts that can be exploited by malicious actors, leading to unintended behavior, financial loss, or security breaches.

6. DDoS Attack (Distributed Denial of Service)

In a DDoS attack, an attacker overwhelms a network, service, or application with a flood of traffic from multiple sources.

7. Blockchain Spamming

refers to the practice of overwhelming a blockchain network with a high volume of transactions or messages, often with the intent to disrupt normal operations . This can lead to increased transaction fees, slower processing times.

8. Selfish Mining

Selfish mining is a strategy used in cryptocurrency mining, particularly in the context of blockchain networks like Bitcoin, where a miner (or group of miners) can attempt to manipulate the system to gain more rewards than they would through honest mining.

9. Routing Attacks : Attackers manipulate internet routing protocols to redirect transactions.

10. Consensus Manipulation: This involves manipulating the consensus process

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

itself, potentially leading to forks or altered transaction histories.

IC.1.3:Designing the architecture of blockchain application

- ✓ **Description of blockchain architecture**

Blockchain architecture is the underlying structure and design of how a blockchain operates. It is a decentralized, distributed system where data is stored in a secure, transparent, and immutable way across a network of participants (nodes). The key feature of this architecture is that no central authority is needed for transaction validation or data storage, ensuring trust and security through cryptographic techniques and consensus mechanisms.

The architecture consists of several interrelated components that work together to ensure the functionality and security of the system. Below is a breakdown of the:

- Components
- Connection
- Instance relation

1. Components of Blockchain Architecture

The key components of blockchain architecture include:

a. Blocks

- Definition: Blocks are units of data that store transaction information. Each block contains a list of transactions, a reference to the previous block (through its cryptographic hash), a timestamp, and a nonce in Proof of Work systems.

- Function: They store the actual data or transaction records in the blockchain.

b. Transactions

- Definition: Transactions are records of actions that occur on the blockchain, such as sending cryptocurrency or executing smart contracts.
- Function: They are the data entries that users make, and these are stored in blocks.

c. Nodes

- Definition: Nodes are the devices (computers) that maintain a copy of the blockchain ledger. Nodes can be full nodes (store the entire blockchain) or light nodes (store part of the blockchain).
- Function: Nodes validate transactions, participate in consensus, and maintain the integrity of the network.

d. Consensus Mechanism

- Definition: The algorithm used by nodes to agree on the state of the blockchain and validate transactions.
- Types: Proof of Work (PoW), Proof of Stake (PoS), Delegated Proof of Stake (DPoS), etc.
- Function: Ensures agreement on the blockchain's state without a central authority, preventing double spending and fraud.

e. Cryptographic Hashing

- Definition: A mathematical algorithm that converts input data into a fixed-size string of characters, typically represented as a hash (e.g., SHA-256).
- Function: Provides security, linking each block to the previous one, and

ensuring the immutability of the blockchain.

f. Smart Contracts : ***Self-executing contracts with terms written into code that automatically execute when predefined conditions are met.***

- Function: Automates transactions, reducing the need for intermediaries.

g. Ledger (Distributed Ledger Technology - DLT) : ***A database that is consensually shared and synchronized across multiple sites, institutions, or geographies.***

- Function: Stores the full history of all transactions in a distributed, tamper-proof manner.

h. Wallets : ***Software or hardware that stores private and public keys used to sign and validate transactions on the blockchain.***

- Function: Allow users to interact with the blockchain by sending, receiving, and managing assets.

i. Security Mechanisms : Security features include cryptography, digital signatures, and encryption techniques that ensure the safety of data and transactions.

- Function: Provides protection against unauthorized access and fraud.

---

2. Connections Between Components

In a blockchain architecture, the components are interconnected in a way that ensures secure and reliable operation. Here's how the components are connected:

a. *Blocks and Transactions*

- Connection: Transactions are collected in a block. Each block contains a list of transactions that have been validated and agreed upon by the network.

b. Blocks and Cryptographic Hashes

- Connection: Each block is linked to the previous block via a cryptographic hash of the previous block's header. This forms a chain of blocks (hence the name "blockchain"), ensuring the immutability of the data.

c. Nodes and Consensus Mechanism

- Connection: Nodes in the network communicate with each other to validate transactions. The consensus mechanism ensures that nodes agree on which block should be added to the blockchain.

d. Transactions and Smart Contracts

- Connection: Some transactions trigger smart contracts. Once the conditions of a smart contract are met, the contract self-executes and creates new transactions that are stored on the blockchain.

e. Nodes and the Distributed Ledger

- Connection: All nodes maintain a copy of the distributed ledger (blockchain). Changes to the blockchain (e.g., new blocks being added) are propagated to all nodes through the consensus process.

f. Wallets and Transactions

- Connection: Users interact with the blockchain via wallets. Wallets generate transactions, which are then signed using the private key and

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

broadcast to the network for validation.

g. Consensus Mechanism and Security

- Connection: The consensus mechanism ensures the security of the blockchain by preventing malicious actors from altering the ledger. It uses cryptographic techniques to validate new blocks and prevent double-spending.

---

3. Instance Relation

Instance relation refers to how the components interact in real-time during blockchain operations. The following explains how these components come together in an actual blockchain network:

a. User Creates a Transaction

- A user uses their wallet to create a transaction (e.g., sending cryptocurrency). This transaction is signed using their private key and then broadcast to the network.

b. Transaction Propagation to Nodes

- The transaction is sent to the nodes in the blockchain network. Full nodes receive the transaction and check its validity (e.g., does the user have enough funds? Is the digital signature valid?).

c. Transaction Inclusion in Block

- Once verified, the transaction is added to a candidate block by a miner or validator. The block also contains the hash of the previous block, ensuring the chain remains intact.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

d. Consensus Mechanism Validates the Block

- The consensus mechanism (e.g., PoW or PoS) ensures that all nodes agree on the validity of the new block. In PoW, miners compete to solve a cryptographic puzzle; in PoS, validators are selected based on their stake in the network.

e. Block Added to Blockchain

- Once the block is validated, it is added to the blockchain, and the new block's hash is shared with all nodes in the network, updating the distributed ledger.

f. Transaction Finalized

- The transaction is now part of the blockchain and is considered finalized after a certain number of confirmations (additional blocks added after it

❖ **Designing system architecture**

- Designing a blockchain-based system involves several key steps, including defining the overall architecture, designing the blockchain network, and developing smart contracts.

❖ Below is a structured approach to each of these components.

✓ **1. Designing Blockchain-Based Systems**

- **A. Define the Use Case**

➢ Clearly define the problem your blockchain application aims to solve.

➢ Identify the users and stakeholders involved in the system.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- Gather functional and non-functional requirements, such as scalability, security, and performance
  - **B. Choose the Type of Blockchain**
- **Public Blockchain:** Open to anyone, fully decentralized (e.g., Bitcoin, Ethereum).
- **Private Blockchain**: Restricted access, controlled by a single organization (e.g., Hyperledger Fabric).
- **Consortium Blockchain**: Controlled by a group of organizations (e.g., R3 Corda).
  - **C. Identify Key Components**
- **Nodes:** Determine the types of nodes required (full nodes, light nodes, miner nodes).
- **User Interface:** Design the front-end application for user interaction.
- **APIs:** Define the APIs that will allow communication between the front-end and blockchain.

✓ **2. Designing the Blockchain Network**

- A. Network Topology
- **Peer-to-Peer (P2P)**: Define how nodes will connect and communicate. A P2P network allows nodes to share information directly.
- **Node Distribution:** Decide on the geographical distribution of nodes to enhance resilience and performance.
  - B. Consensus Mechanism
- Selection: Choose a consensus algorithm that fits the use case (e.g., Proof

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

of Work, Proof of Stake, Byzantine Fault Tolerance).

- Parameters: Define the parameters for the consensus mechanism, such as block time, reward distribution, and penalties for dishonest behavior.

  - C. Security Measures

- Encryption: Implement cryptographic techniques to secure data and transactions.
- Access Control: Establish role-based access control (RBAC) or other mechanisms to restrict access to sensitive operations.

  D. Monitoring and Maintenance

- Monitoring Tools: Set up tools to monitor network performance, node health, and transaction throughput.
- Update Strategy: Define a strategy for regular updates and maintenance of the network.

✓ **3. Designing Smart Contracts**

- A. Define the Business Logic

- Clearly define the business rules and logic that the smart contract will enforce.
- Create flowcharts or diagrams to visualize how the smart contract will operate.

- B. Development Environment

- Select a programming language for smart contract development (e.g., Solidity for Ethereum, Chaincode for Hyperledger Fabric).
- Use frameworks and tools that facilitate smart contract development

(e.g., Truffle for Ethereum).

- C. Smart Contract Structure

- Define the data structures needed to store state variables and manage user data.
- Implement functions for key operations, including:
- **Creation:** Functions to create new instances (e.g., creating a new asset).
- **Modification:** Functions to update state (e.g., transfer ownership).
- **Validation:** Functions to validate conditions before executing critical operations.

- D. Testing and Auditing

- Unit Testing: Write unit tests to ensure the smart contract functions correctly.
- Integration Testing: Test the interaction between the smart contract and other components.
- Security Audits: Conduct thorough audits to identify vulnerabilities and ensure the contract is secure.

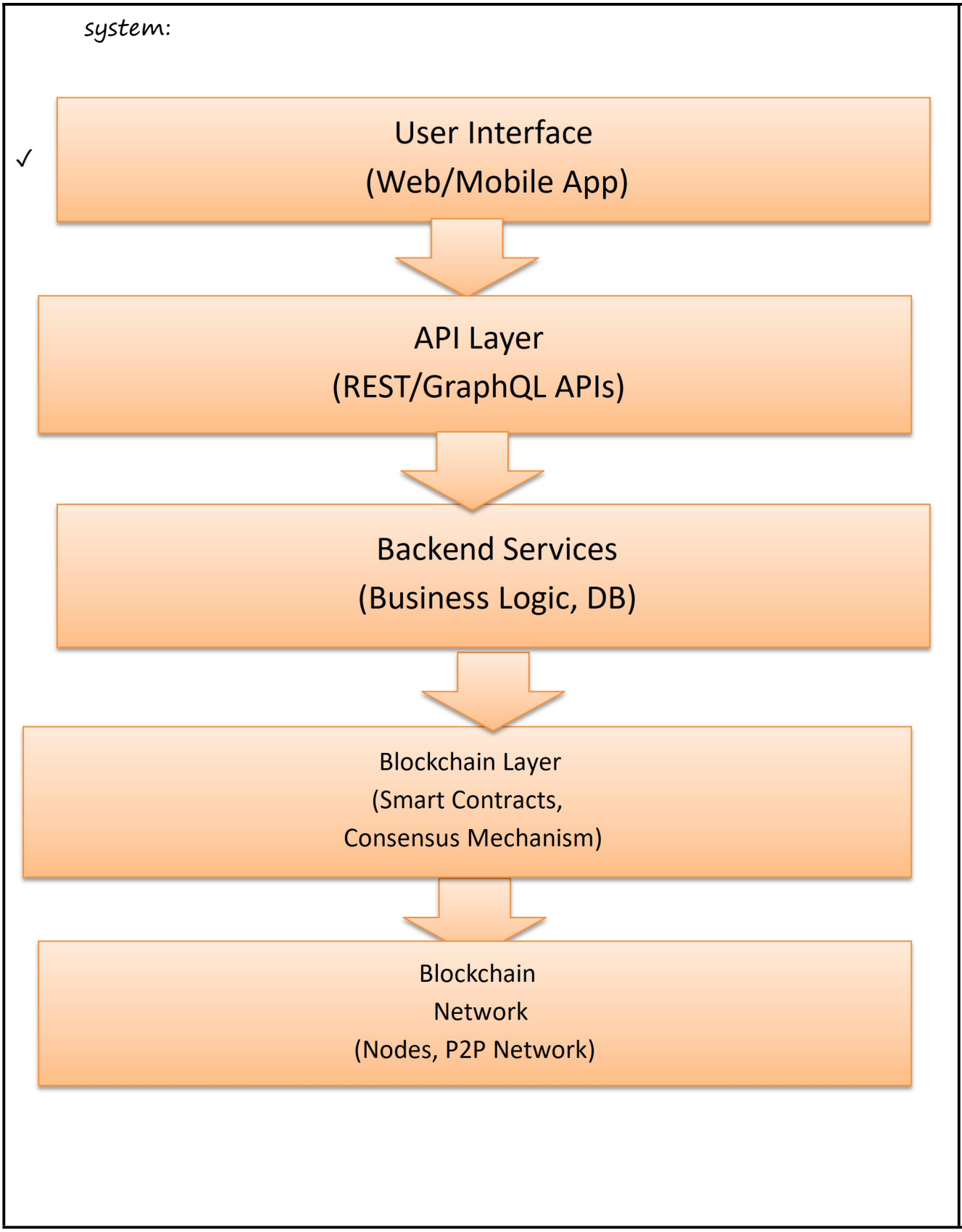
- E. Deployment

- Deployment Strategy: Define how and when the smart contract will be deployed to the blockchain.
- Version Control: Implement a versioning system to manage updates to the smart contract.
- Example Architecture Overview

  Here's a simplified overview of the architecture for a blockchain-based

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

system:



Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- **Drawing blockchain architecture**

- Designing a blockchain architecture involves several steps, including identifying the use case, third-party integrations, consensus mechanisms, platforms, and the overall architecture design.

- Below is a detailed breakdown of each step, along with a visual representation of a typical blockchain architecture.

- ✓ **1. Identify the Use Case**

  - Use Case Example: Supply Chain Management
  - Problem Statement: Improve transparency and traceability in the supply chain by tracking products from manufacturers to consumers.
  - Stakeholders: Manufacturers, suppliers, distributors, retailers, and consumers.

- ✓ **2. Identify Third-Party Integration**

- Third-Party Integrations:

  - Payment Processors: Integrate with services like PayPal, Stripe, or cryptocurrency payment gateways for transactions.
  - Identity Verification Services: Use KYC (Know Your Customer) services to verify the identities of participants.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- IoT Devices: Integrate IoT sensors to automatically update the blockchain with real-time data (e.g., temperature, location).
- Data Oracles: Use oracles to bring off-chain data onto the blockchain for smart contract execution.

✓ **3. Identify the Consensus Mechanism**

➢ Consensus Mechanism:

- Type: Proof of Stake (PoS)
- Reason: PoS is energy-efficient and suitable for a permissioned blockchain where stakeholders are known and can be trusted to some extent.

✓ **4. Identify the Platform**

➢ Blockchain Platform:

- Platform Choice: Hyperledger Fabric
- Reason: Hyperledger Fabric provides a modular architecture, supports private transactions, and allows for permissioned access, making it suitable for enterprise applications like supply chain management.

✓ **5. Design the Blockchain Instance**

➢ Blockchain Instance Design:

- Node Types:
- **Orderer Nodes:** Responsible for transaction ordering and consensus.
- **Peer Nodes:** Maintain the ledger and execute smart contracts.
- **Client Nodes:** Interact with the network through the application

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

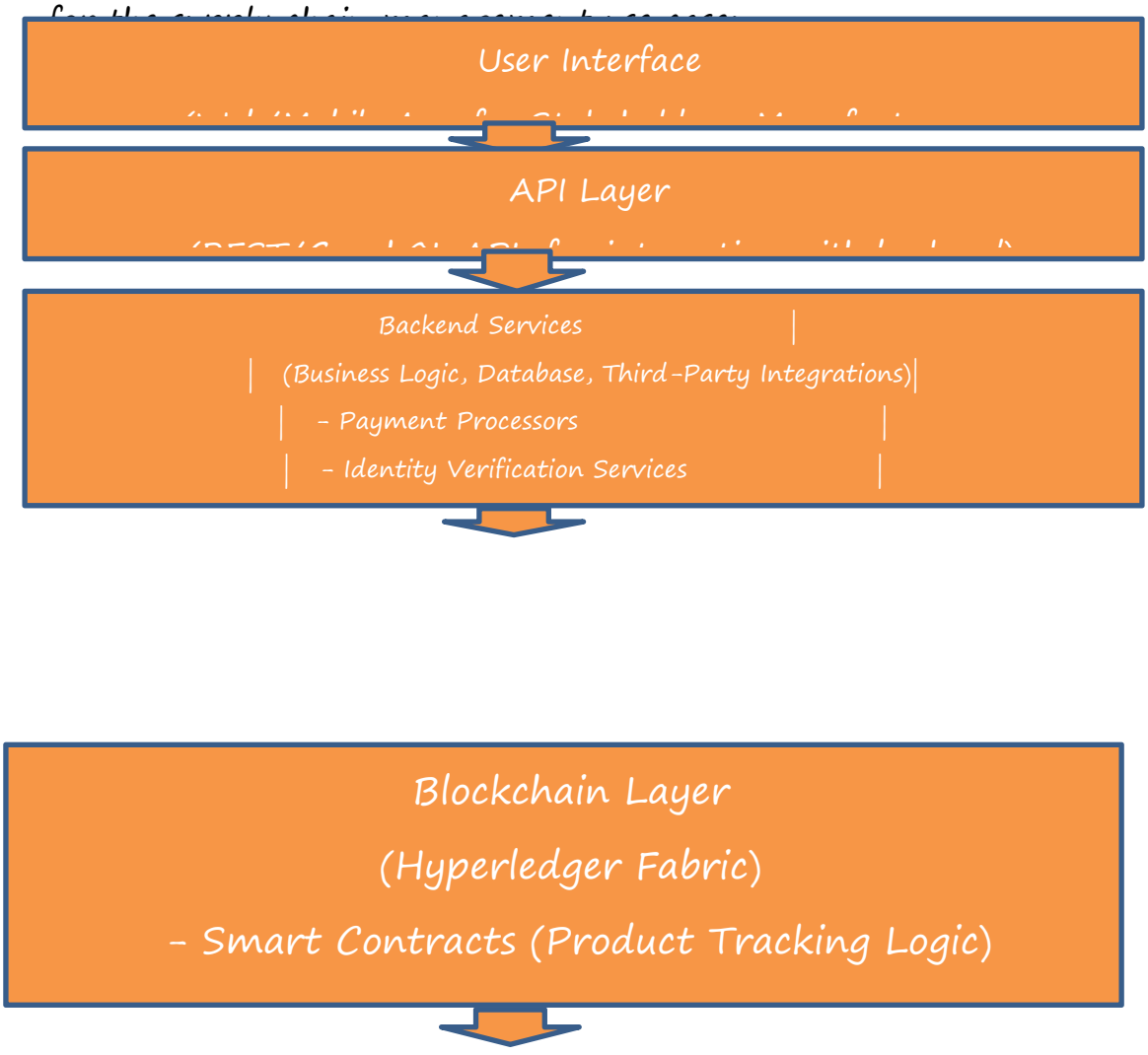
interface.

- **Smart Contracts**: Define the business logic for tracking product movements, ownership transfers, and compliance checks.

✓ **6. Design the Architecture**

➢ Blockchain Architecture Diagram:

❖ Here's a simplified visual representation of the blockchain architecture



Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Blockchain Network
(Nodes: Orderer, Peer, Client)
- Orderer Nodes (Transaction Ordering)
- Peer Nodes (Ledger Maintenance, Smart Contract Execution)

- This structured approach to designing blockchain architecture includes identifying the use case, third-party integrations, consensus mechanisms, and the platform to be used.
- The architecture diagram illustrates how these components interact with each other, providing a clear overview of the system's design.
- By following these steps, you can create a robust blockchain solution tailored to specific business needs.

REVIEW / SUMMARY TO:

Designing Blockchain Application Architecture

- Blockchain Architecture:
  - ✓ The overall structure of how a blockchain system is built.
  - ✓ **Components:**
    - ➢ The different parts that make up the blockchain, like nodes (computers), the ledger (where transactions are recorded), and protocols (rules for how it works).
  - ✓ **Connection:**

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- How these parts communicate with each other, ensuring they work together smoothly.

✓ **Instance Relation:**

- How different parts of the system relate to each other, like how a user interacts with the blockchain.

❖ **Designing System Architecture:**

✓ Design Blockchain-Based Systems:

- Creating systems that use blockchain technology for various applications, like finance or supply chain.

✓ Designing the Blockchain Network:

- Planning how the network of computers (nodes) will be set up, including how they will connect and share information.

✓ Design Smart Contracts:

- Writing rules and agreements that automatically execute when certain conditions are met, like a digital contract that runs on the blockchain.

❑ Conclusion

✓ Designing a blockchain application involves planning its structure, how its parts connect, and how it functions. This includes creating the network, systems, and smart contracts that will make it work effectively

Drawing Blockchain Architecture

❖ Identify the Use Case:

- Determine what problem the blockchain will solve or what application it

will serve (e.g., digital payments, supply chain tracking).

- ✓ Identify Third-Party Integration:
  - ➢ Figure out if the blockchain needs to work with other systems or services, like payment processors or identity verification services.
- ✓ **Identify the Consensus Mechanism:**
  - ➢ Decide how the network will agree on which transactions are valid. Common methods include Proof of Work (like Bitcoin) or Proof of Stake.
- ✓ **Identify the Platform:**
  - ➢ Choose the technology or framework that will be used to build the blockchain, such as Ethereum, Hyperledger, or others.
- ✓ **Design the Blockchain Instance:**
  - ➢ Plan how the specific blockchain will be set up, including the number of nodes, their roles, and how they will communicate.
- ✓ **Design the Architecture:**
  - ➢ Create a visual representation of the entire system, showing how all the components fit together, including nodes, smart contracts, and user interfaces.

❑ **Conclusion**

- ✓ Drawing the architecture of a blockchain application involves understanding the problem it will solve, how it will connect with other services, and how it will operate. This helps create a clear plan for building the blockchain system.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Learning outcome 2: Apply Solidity Basics	Learning hours: 20 hours
Indicative content	

Ind.c:2.1 Preparation of environment

- ✓ Description of key terms

Solidity is a high-level programming language designed for writing smart contracts on blockchain platforms, primarily Ethereum.

- ✓ It is statically typed, supports inheritance, libraries, and complex user-defined types.

❖ 1. Solidity Basics

➢ **A. Setting Up the Environment**

✓ To start writing Solidity smart contracts, you can use several development environments, such as:

- **Remix IDE**(*Integrated Development Environment*): An online IDE specifically for Solidity development.
- **Truffle Suite**: A development framework for Ethereum.
- **Hardhat**: A development environment for compiling, deploying, testing, and debugging Ethereum software.

➢ B. Solidity Syntax

// **SPDX-License-Identifier:** MIT :This is a license identifier that specifies the license under which the contract is published.

MIT(**Massachusetts Institute of Technology.**): means the code is licensed under the permissive MIT license.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- 1.Version Pragma: Specifies the version of Solidity to be used.

❖ pragma solidity ^0.8.0;

- 2.Contract

➢ A contract in Solidity is similar to a class in object-oriented programming.

❖ contract MyContract {

    // State variables and functions go here

}

- 3. State Variables: Variables that hold the state of the contract.

❖ uint public myNumber; // Unsigned integer

❖ string public myString; // String

❖ address public owner; // Ethereum address

- 4.Functions: Functions define the behavior of the contract.

❖ function setMyNumber(uint _number) public {

    myNumber = _number;

}

- 5.Modifiers: Used to change the behavior of functions. Commonly used for access control.

❖ modifier onlyOwner() {

require(msg.sender == owner, "Not the contract owner");

    _;

}

- 6.Events: Used to log information on the blockchain. External listeners can subscribe to these events.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- event NumberSet(uint indexed newNumber);

2. Data Types

Definition: Solidity has various data types to define the type of data a variable can hold.

Examples:

Integers: uint (unsigned) and int (signed).

Booleans: bool (true or false).

Addresses: address (stores Ethereum addresses).

Strings: string (text).

Bytes: bytes, bytes32, etc., for fixed-length byte arrays.

3. Variables

Definition: Variables are used to store data that can be used and manipulated within the contract.

Types:

State Variables: Stored on the blockchain.

Local Variables: Temporary variables used within functions.

Global Variables: Provide information about the blockchain (e.g., msg.sender).

4. Identifiers

Definition: Names given to variables, functions, contracts, etc. Identifiers must be unique and descriptive.

Example:

```
uint totalSupply;
function calculateSum() { ... }
```

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

5. Arrays

Definition: Lists of elements of the same data type, either fixed or dynamic in size.

Examples:

```
uint[] public dynamicArray;
uint[5] public fixedArray;
```

6. Struct

Definition: A way to define custom data types by grouping variables of different types.

Example:

```
struct Person {
    string name;
    uint age;
}
Person public person = Person("Alice", 30);
```

7. Functions

Definition: Blocks of code that perform specific tasks and can be called to execute certain operations.

Example:

```
function add(uint a, uint b) public pure returns (uint) {
    return a + b;
}
```

public: Anyone can call this function.

pure: The function does not read or modify state variables.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

8. Control Structures

Definition: These are decision-making structures used to control the flow of execution (similar to other programming languages).

Examples:

If/Else:

```
if (x > 10) {
    return true;
} else {
    return false;
}
```

For Loop:

```
for (uint i = 0; i < 10; i++) {
    // Do something
}
```

9. State Variables

Definition: Variables that are permanently stored on the blockchain.

Example:

uint public balance = 1000;

The value of balance will be stored on-chain.

10. Modifiers (Conditions)

Definition: Reusable pieces of code that can be attached to functions to modify their behavior.

Example:

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

```
modifier onlyOwner() {
    require(msg.sender == owner, "Not authorized");
    _;
}
function withdraw() public onlyOwner { ... }
```

The function withdraw can only be called by the contract owner.

11. Smart Contract

Definition: A self-executing contract with the terms of the agreement directly written into code. They automatically enforce and execute agreements when conditions are met.

Example:

```
contract SimpleStorage {
    uint data;
    function set(uint x) public {
        data = x;
    }
    function get() public view returns (uint) {
        return data;
    }
}
```

12. Visibility and Access Control

Definition: Visibility determines who can access a function or state variable.

public: Accessible by anyone.

internal: Only accessible within the contract or derived contracts.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

private: Only accessible within the contract where it is defined.

external: Can only be called from outside the contract.

13. Ethereum

Definition: A decentralized blockchain platform that allows developers to deploy smart contracts and build decentralized applications (DApps).

Currency: Uses the native cryptocurrency Ether (ETH) for transactions and computational costs (gas fees).

14. Ethereum Virtual Machine (EVM)

Definition: The runtime environment for executing smart contracts on the Ethereum blockchain. It handles code execution, state changes, and ensures security and isolation.

Role: Every Ethereum node runs an instance of the EVM, which interprets and executes the bytecode of smart contracts.

These are the foundational concepts of Solidity and Ethereum, crucial for anyone looking to develop on the Ethereum blockchain.

- ✓ **Set up solidity environment**

- Installing Code editor (remix, visual studio code)
- Installing node.js and npm (Node Package Manager) for package management.
- Installing Solidity compiler (solc) and Ethereum development tools (e.g., Truffle, Hardhat).

Step 1: Install a Code Editor

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

A code editor is where you'll write your Solidity smart contracts. There are two primary options:

Option 1: Remix IDE (Browser-Based)

- **What is Remix?**

- Remix is an online tool specifically designed for Solidity development.
- It includes a Solidity compiler, deployment tools, and debugging features.
- It's beginner-friendly and requires no local installation.

- **How to Use Remix**:

- Go to Remix IDE.
- Start coding directly in the browser.
- It provides a file explorer, smart contract compilation, and test deployment tools.

- **Advantages**:

    - No setup required.
    - Suitable for quick prototyping and small projects.

Option 2: Visual Studio Code (VS Code)

- **What is VS Code?**

- A popular code editor for various programming languages, including Solidity.
- Offers more customization and flexibility for advanced projects.

- **How to Install VS Code**:

- Download from Visual Studio Code's website.
- Install it on your system.

- **Install Solidity Plugin for VS Code**:

- Open VS Code.
- Go to the Extensions Marketplace (Ctrl+Shift+X).
- Search for "Solidity" and install the extension.
- This plugin provides features like syntax highlighting, code linting, and Solidity snippets.

- **Why Use VS Code?**

- Ideal for larger projects.

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- Supports version control (Git) and integration with advanced tools.

Step 2: Install Node.js and npm

- **Why Node.js?**

- Node.js enables JavaScript runtime outside the browser, which is essential for using Ethereum development tools.
- **npm** (Node Package Manager) is used to manage the dependencies and tools required for Solidity development.

- **How to Install Node.js**:

- Download from the Node.js official website.
- Choose the **LTS version** for stability.
- Install Node.js, which automatically includes npm.

- **Verify Installation**: Open your terminal or command prompt and run:

- `node -v`
- `npm -v`

  This should display the installed versions of Node.js and npm.

Step 3: Install Solidity Compiler (solc)

The Solidity compiler (`solc`) converts Solidity code into bytecode that the Ethereum Virtual Machine (EVM) can execute.

Install solc Using npm:

1. Open your terminal.
2. Run:

```
3. npm install -g solc
```

Verify Installation:

Run:

```
solcjs --version
```

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

This will display the installed version of the Solidity compiler.

Alternative: Use Remix:

If you're using Remix, it has a built-in Solidity compiler, so you can skip this step for browser-based development.

Step 4: Install Ethereum Development Tools

Ethereum development tools simplify tasks like testing, debugging, and deploying smart contracts. Two popular tools are **Truffle** and **Hardhat**.

Option A: Truffle

- **What is Truffle?**

➢ A framework for developing, testing, and deploying Solidity contracts.
➢ Includes a built-in testing environment and migration system.

- **Install Truffle**:

```
- npm install -g truffle
```

- **Verify Installation**:

```
- truffle version
```

- **Create a New Truffle Project**:

```
- mkdir my-truffle-project
- cd my-truffle-project
- truffle init
```

Option B: Hardhat

- **What is Hardhat?**

➢ A modern framework for Ethereum development.
➢ Offers flexibility for advanced debugging, contract testing, and deployment.

- **Set Up a Hardhat Project**:

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

- **Create a project folder:**

```
mkdir my-hardhat-project
cd my-hardhat-project
npm init -y
```

- **Install Hardhat:**

```
npm install --save-dev hardhat
```

- **Initialize Hardhat:**

```
npx hardhat
```

Follow the prompts to set up the project.

Step 5: (Optional) Install Ganache for Local Testing

- **What is Ganache?**

- A personal blockchain used for testing Solidity contracts locally.
- It provides accounts with preloaded Ether for testing purposes.

- **Install Ganache**:

```
npm install -g ganache
```

- **Start Ganache**:

```
ganache
```

This will start a local blockchain.

Testing the Environment

1. Write a simple Solidity contract (e.g., `HelloWorld.sol`).
2. Compile it using:
   - Remix (online).
   - solc (command-line).
   - Truffle or Hardhat (framework-based).
3. Deploy it to a test blockchain (Ganache or Hardhat Network).

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

4. Interact with the deployed contract using tools like:
    - Truffle Console
    - Hardhat scripts
    - Web3.js or ethers.js libraries in Node.js.

Recap of Tools

Tool	Purpose
Remix	Online editor for quick Solidity coding.
VS Code	Advanced code editor with Solidity support.
Node.js	Runtime environment for JavaScript-based tools.
solc	Solidity compiler for local compilation.
Truffle	Framework for testing and deploying contracts.
Hardhat	Modern development framework with debugging tools.
Ganache	Personal blockchain for local testing.

Ind.cont 2.2: Applying solidity concepts

- ✓ Data types and variables
- ✓ Use of functions
- ✓ Control structures
- ✓ arrays and structs
- ✓ Events and logging
- ✓ Error handling

1. Data Types and Variables

In Solidity, **data types** define the kind of values that variables can hold. Solidity supports several data types grouped into **value types** and **reference types**.

Value Types:

- **`uint`**: Unsigned integers (non-negative whole numbers). Example: `uint256` (256-bit

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

unsigned integer).
- **`int`**: Signed integers (can be positive or negative). Example: `int256`.
- **`bool`**: Boolean type, holds `true` or `false`.
- **`address`**: Represents an Ethereum address (20 bytes).

Reference Types:

- **Arrays**: Hold collections of elements of the same type.
- **Structs**: Combine multiple data fields into a single entity.

Declaring Variables:

Variables can be declared as:

- **State variables**: Stored on the blockchain and persist across function calls.
- **Local variables**: Exist only during function execution.
- **Global variables**: Provided by Solidity (e.g., `msg.sender`, `block.timestamp`).

Example:

```
contract DataTypes {
    uint256 public age = 30;    // State variable
    bool public isActive = true;
    address public owner;
    constructor() {
        owner = msg.sender;     // Global variable to store the deployer's address
    }
    function setAge(uint256 _age) public {
        age = _age;             // Modify the state variable
    }
    function calculateSum(uint256 a, uint256 b) public pure returns (uint256) {
        uint256 sum = a + b;    // Local variable
        return sum;
    }
}
```

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

2. Use of Functions

Functions define actions in Solidity contracts. They can:

1. Modify the state.
2. Perform calculations.
3. Interact with other contracts.

Function Types:

- **`public`**: Accessible from both within and outside the contract.
- **`private`**: Accessible only within the contract.
- **`external`**: Called only from outside the contract.
- **`internal`**: Called within the contract or derived contracts.

Modifiers:

- **`view`**: Used when the function only reads data.
- **`pure`**: Used when the function doesn't interact with state variables or blockchain data.

Example:

```
contract Functions {
    uint256 public counter;
    // Function to increment the counter
    function increment() public {
        counter += 1;   // Modifies state variable
    }
    // Pure function to add two numbers
    function add(uint256 a, uint256 b) public pure returns (uint256) {
        return a + b;
    }
    // View function to retrieve counter value
    function getCounter() public view returns (uint256) {
        return counter;
    }
}
```

3. Control Structures

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Control structures allow for conditional execution and loops.

Conditional Statements:

- **`if` and `else`**: Execute code based on conditions.

Loops:

- **`for`**, **`while`**, and **`do-while`** loops iterate over a block of code.

Example:

```
contract ControlStructures {
    // Determine if a number is even
    function isEven(uint256 number) public pure returns (bool) {
        if (number % 2 == 0) {
            return true;
        } else {
            return false;
        }
    }
    // Sum numbers in an array
    function sumArray(uint256[] memory numbers) public pure returns (uint256) {
        uint256 sum = 0;
        for (uint256 i = 0; i < numbers.length; i++) {
            sum += numbers[i];
        }
        return sum;
    }
}
```

4. Arrays and Structs

Arrays:

Arrays store collections of data of the same type.

- **Fixed-size array**: Defined with a specific size.
- **Dynamic array**: Can grow or shrink.

Structs:

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

Structs group multiple data fields into a single entity.

Example:

```
contract ArraysAndStructs {
    // Dynamic array
    uint256[] public dynamicArray;

    // Add element to the dynamic array
    function addToArray(uint256 number) public {
        dynamicArray.push(number);
    }

    // Struct to store person details
    struct Person {
        string name;
        uint256 age;
    }
    Person[] public people;
    // Add a person to the array
    function addPerson(string memory _name, uint256 _age) public {
        people.push(Person(_name, _age));
    }
}
```

5. Events and Logging

Events provide a way to log data on the blockchain. Logs are not stored in contract storage but are accessible to external applications like dApps.

Usage of Events:

- Declared using the `event` keyword.
- Triggered using the `emit` keyword.

Example:

```
contract Events {
    // Declare an event
    event Transfer(address indexed from, address indexed to, uint256 amount);
```

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

```
    // Emit the event in a function
    function transfer(address to, uint256 amount) public {
        emit Transfer(msg.sender, to, amount); // Logs the transfer details
    }
}
```

6. Error Handling

Error handling ensures the contract behaves as expected by reverting invalid operations. Key mechanisms:

1. **`require`**: Validates user inputs and conditions.
2. **`revert`**: Triggers an error with an optional message.
3. **`assert`**: Checks invariants (e.g., internal consistency). Should only be used for debugging critical issues.

Example:

```
contract ErrorHandling {
    uint256 public balance;
    // Function to deposit Ether
    function deposit() public payable {
        require(msg.value > 0, "Must send Ether to deposit");
        balance += msg.value;
    }
    // Function to withdraw Ether
    function withdraw(uint256 amount) public {
        require(amount <= balance, "Insufficient balance");
        balance -= amount;
        payable(msg.sender).transfer(amount);
    }
    // Use assert to check invariants
    function checkInvariant() public view {
        assert(balance >= 0); // The balance should never be negative
    }
}
```

A Complete Example: Bank Contract

Here's a practical example that combines all the above concepts:

Link: https://www.tutorialspoint.com/solidity/solidity_functions.htm

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.7.0 <0.9.0;
contract Balance{
    int bal;
    // constructor to initialize the balance
    constructor(){
        bal = 10;
    }
    // function to return amaount of this contact
    function getBalance() public  view  returns (int){
        return bal;
    }
// function to withdraw a specific amount from balance
function withdraw(int amt) public {
    require(amt <= bal, "insuffient balance ");
    bal = bal - amt;
}
// function to depositi a specific amount into balance
function deposit(int amt) public {
    bal = bal + amt;
    // bal += amt;
}
}
```

Explanation of the Bank Contract:

1. **Data Types**:
    - `mapping` stores balances linked to user addresses.
2. **Functions**:
    - `deposit` allows users to send Ether.
    - `withdraw` lets users retrieve funds.
    - `getBalance` provides account details.
3. **Events**:
    - `Deposit` and `Withdrawal` log transactions.
4. **Error Handling**:
    - `require` ensures valid deposits and prevents overdrawing.