

Laboratory Manual

DEPARTMENT OF INFORMATION TECHNOLOGY

PROGRAM - B.E
FUNDAMENTALS OF BLOCKCHAIN TECHNOLOGY (20ITE21)
SEMESTER - VII
R-20 Regulation

Prepared by:

Verified by:



CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE,
Accredited by NAAC with A++ Grade and Programs Accredited by NBA)

Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana

www.cbit.ac.in



CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE,
Accredited by NAAC with A++ Grade and Programs Accredited by NBA)
Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana
www.cbit.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

Name of the Lab Course:

FUNDAMENTALS OF BLOCKCHAIN TECHNOLOGY LAB (2OITE21)

INDEX

Item	Page No
Institute Vision, Institute Mission, Quality Policy, Department Vision and Department Mission	4
Program Educational Objectives (PEOs), Program Outcomes (POs) and Program Specific Outcomes (PSOs)	5
Syllabus	9
Course Introduction	10
Assessment Procedure of CIE and SEE	12-15
General Instructions for Laboratory Classes	16
Concluding Remarks / Gap Analysis	56

LABORATORY / PRACTICAL

Exp No	Laboratory / Practical	CO	BTL	Page No
1	Understanding Blockchain Foundations: Elements of Distributed Computing, Elements of Cryptography, Digital Signature.	1	L2	12-15
2	Getting familiar with the Ethereum platform and Ethereum virtual machine.	1	L2	19-22
3	Introduction to solidity program structure, compilation and deployment environment.	2	L2	22-24
4	Creating and deploying the simple smart contract to store and get "Hello World", on the Blockchain network.	3	L6	24-27
5	Develop a smart contract to create a function setter and getter to set and get a value.	3	L3	27-30
6	Develop Solidity contracts to illustrate inheritance and polymorphism.	3	L3	30-34
7	Familiarize with the working of Remix Ethereum tool.	4	L2	35-39
8	Design a smart contract on Remix Ethereum to print the array of integers and its length.	4	L3	39-43

9	Setup a Simple Ethereum wallet and use it to send and receive Ethers	3	L3	43-46
10	Hyperledger Fabric Demo	5	L4	47-49
Experiments Beyond curriculum (Open ended / Structured enquiry)				
1	Develop a multi-signature wallet smart contract. This contract would require multiple parties to approve a transaction before it can be executed.	3	L3	49-51
2	Build a simple decentralized application using Ethereum as the backend.	3	L5	51-52
3	Design and implement a crowdfunding smart contract where users can contribute Ether to a project, and the funds are released only when a predefined goal is reached.	3	L3	52-54
4	Develop a web interface that interacts with your deployed smart contracts using the Web3.js library.	4	L3	54-57

Note:

- Minimum 12 experiments should be included in every course other than experiments beyond syllabus.
- Add project/ case studies/ student defined experiments for the courses having experiments less than 12 in syllabus.
- Include Minimum 2 experiments beyond prescribed experiments meeting industry problems to challenge student learning.



CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE,
Accredited by NAAC with A++ Grade and Programs Accredited by NBA)
Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana
www.cbit.ac.in

Vision of Institute

To be the Centre of Excellence in Technical Education and Research.

Mission of Institute

To address the Emerging needs through Quality Technical Education
and Advanced Research.

Quality Policy

CBIT imparts value based Technical Education and Training to meet
the requirements of students, Industry, Trade/ Profession, Research
and Development Organizations for Self-sustained growth of Society.



CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE,
Accredited by NAAC with A++ Grade and Programs Accredited by NBA)
Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana
www.cbit.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

Vision of the Department

To be a center of excellence in the field of information technology yields pioneers and research experts who can contribute for the socio-economic development of the nation.

Mission of the Department

- To impart state-of-the-art value-based education in the field of Information Technology.
- To collaborate with industries and research organizations and excel in the emerging areas of research.
- To imbibe social responsibility in students.
- To motivate students to be trend setters and technopreneurs

Program Educational Objectives (PEOs)

- Analyze and provide solutions for real world problems using state-of-the-art engineering, mathematics, computing knowledge, and emerging technologies.

Exhibit professional leadership qualities and excel in interdisciplinary domains.

Demonstrate human values, professional ethics, skills and zeal for lifelong learning.

- Contribute to the research community and develop solutions to meet the needs of public and private sectors. Work in emerging areas of research and develop solutions to meet the needs of public and private sectors.



CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE,
Accredited by NAAC with A++ Grade and Programs Accredited by NBA)
Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana
www.cbit.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

Program Outcomes (POs)

PO1. Engineering Knowledge:

Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.

PO2. Problem analysis:

Identify, formulate, review, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/development of solutions:

Design solutions for complex engineering problems and design system components that meet the specified needs with appropriate consideration for the public health, safety, and cultural, societal, and environmental considerations.

PO4. Conduct investigations of complex problems:

Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. Modern tool usage:

Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6. The engineer and society:

Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional

engineering practice.

PO7. Environment and sustainability:

Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics:

Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and team work:

Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication:

Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project management and finance:

Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-long learning:

Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

After successful completion of the program, students will be able to:

PSO1. Contribute to the growth of the nation by providing IT enabled solutions.

PSO2. Develop professional skills in the thrust areas like Computer Networks, Image ssing, Data Mining, Internet of Things, Cloud Computing and Information Security.

PSO3. Pursue higher studies in specializations like Artificial Intelligence, Data Science, Security and Software Engineering in reputed Universities.

Instruction	0L-0T-2P
Duration of SEE	3 Hours
SEE	50 Marks
CIE	50 Marks
Credits	1

Course Objectives: The objectives of this course are to:

1. To familiarize the basic concepts of blockchain.
2. To provide the significance of the Ethereum blockchain.
3. To introduce solidity programming for developing blockchain applications
4. To explore Remix Tool for developing smart contracts.
5. To explore the features of Hyper ledger Fabric .

Course Outcomes: After completion of course, students would be able to:

1. Explore the working of blockchain fundamentals such as cryptography and distributed computing.
2. Implement smart contract on the Ethereum blockchain.
3. Build smart contracts using Solidity programming language
4. Write smart contracts using the Remix tool.
5. Acquire thorough knowledge of Hyperledger fabric.

Laboratory / Practical:

1. Understanding Blockchain Foundations: Elements of Distributed Computing, Elements of Cryptography, Digital Signature.
2. Getting familiar with the Ethereum platform and Ethereum virtual machine.
3. Introduction to solidity program structure, compilation and deployment environment.
4. Creating and deploying the simple smart contract to store and get "Hello World", on the Blockchain network.
5. Develop a smart contract to create a function setter and getter to set and get a value.
6. Develop Solidity contracts to illustrate inheritance and polymorphism.
7. Familiarize with the working of Remix Ethereum tool.
8. Design a smart contract on Remix Ethereum to print the array of integers and its length.
9. Setup a Simple Ethereum wallet and use it to send and receive Ethers.
10. Hyperledger Fabric Demo.

Text Book:

1. Imran Bashir, "Mastering Blockchain", 2 nd Edition, Packt Publishers, 2018.
2. Melanie Swan, "BlockChain: Blueprint for a New Economy", 1 st Edition, O'Reilly, 2018.

Online Resources:

1. <https://www.coursera.org/learn/blockchain-basics>
2. <https://blockchainlab.com/>
3. <https://www.web3labs.com/>
4. <https://www.blockchainresearchlab.org/>

Course Articulation Matrix:

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	3	2	2	2							1
CO2	3	3	2	2	3		2				1	1
CO3	3	2	2	3	2	2	1					
CO4	2	2	2	3	2	1	1				1	
CO5	3	2	3	3	3	3	1					
Avg	2.8	2.4	2.2	2.6	2.4	2	1.25				1	1

COURSE INTRODUCTION

Name of the Lab Course:

FUNDAMENTALS OF BLOCKCHAIN TECHNOLOGY LAB (20ITE21)

Blockchain technology has emerged as a transformative force that is revolutionizing industries and reshaping the way we think about data, transactions, and trust. At its core, blockchain is a decentralized and distributed digital ledger that records transactions in a secure, transparent, and tamper-resistant manner. This technology has far-reaching implications for sectors ranging from finance and supply chain management to healthcare and beyond.

Decentralization: Unlike traditional centralized systems, where a single entity holds control, blockchain operates on a decentralized network of computers, known as nodes. Each participant in the network maintains a copy of the entire blockchain, ensuring that no single point of failure exists.

Blocks and Chains: A blockchain consists of a series of "blocks," each containing a set of transactions. These blocks are linked together in chronological order, forming a "chain." The chaining of blocks using cryptographic hashes ensures the integrity and immutability of the data.

Consensus Mechanisms: To validate transactions and achieve agreement on the state of the blockchain, consensus mechanisms are employed. Mechanisms like Proof of Work (PoW) and Proof of Stake (PoS) enable students to reach a consensus without relying on a central authority.

Cryptographic Security: Cryptography is at the heart of blockchain security. Transactions are securely signed using cryptographic keys, and each block's hash is based on the previous block's hash, creating a chain that is resistant to tampering.

Transparency and Immutability: Once data is recorded on the blockchain, it becomes extremely difficult to alter. This immutability ensures that the historical record of transactions is transparent, traceable, and auditable.

Smart Contracts: Smart contracts are self-executing contracts with predefined rules encoded directly onto the blockchain. They automatically execute and enforce terms when conditions are met, reducing the need for intermediaries and enhancing efficiency.

Applications:

Financial Services: Blockchain enables secure and fast cross-border payments, reduces settlement times, and opens opportunities for new financial instruments like cryptocurrencies and stablecoins.

Supply Chain Management: Tracking the origin and journey of products becomes more transparent, reducing fraud and ensuring the authenticity of goods.

Healthcare: Patient data can be securely stored, shared, and accessed by authorized parties, leading to more coordinated care and accurate medical records.

Identity Verification: Blockchain can provide a secure and decentralized way to verify identities, reducing the risks associated with centralized identity databases.

Voting Systems: Blockchain-based voting systems enhance transparency and security in elections, reducing the potential for fraud and manipulation.

Digital Ownership (NFTs): Non-fungible tokens (NFTs) on the blockchain authenticate ownership of digital assets, revolutionizing art, collectibles, and digital content ownership.

Applications of Fundamentals of Blockchain Technology Lab using Remix IDE and Solidity:

1. Creating a Basic Smart Contract

Writing a simple Solidity smart contract.

Compiling the contract and deploying it on the test network.

Developing Practical Blockchain Applications

2. Decentralized Voting System

Designing a secure and transparent voting smart contract.

Implementing voter registration, vote casting, and tallying.

3. Supply Chain Management

Building a supply chain tracking smart contract.

Tracking the movement of goods and ensuring authenticity.

4. Decentralized Finance (DeFi) Application

Developing a basic lending or borrowing protocol.

Creating a token, setting interest rates, and managing loans.

5. Non-Fungible Tokens (NFTs)

Exploring the concept of NFTs and their use cases.

Developing a simple NFT smart contract for digital asset ownership.

ASSESSMENT PROCEDURE AND AWARD OF CIE

MARKS

Following is the subdivision for the internal marks (50) of the Lab:

- (i) 20 marks for the lab internal tests

Two tests are to be conducted i.e one test after 1st cycle of experiments and second test after the second cycle. Average of two tests marks put together should be consider (20 maximum)

- (ii) 30 marks for CIE

For the CIE 30 marks will be awarded based on the rubrics provided below.

This Rubrics are general guideline. Based on the lab type (programming or hardware) and complexity of the course Rubrics can be customized by the department in tune to program and course offered. Performance Indicators of the Rubrics also can be changed by the departments/Program based on the need.

RUBRICS

S. No	Parameter	Descriptors and Score				
		Outstanding	Good	Fair	Poor	Very Poor
1	Pre-Experiment Preparation work 5 M	Well prepared for the experimentation with a clear specifications, plan/design and additional information (5M)	prepared for the experimentation with clear specifications and plan/design (4M)	Adequately prepared for the experimentation without clear specifications and plan/design (3M)	Minimal preparation and without clear specifications and plan/design (2M)	Lacks preparation and without clear specifications and plan/design (1 M)
2	Experimentation (Problem Solving, Methodology of Conduction) 10 M	Student conducts experiment with all possible test cases in an optimized fashion.(10M)	Student solves the problem and conducts experiments with all possible test cases (8M)	Student solves the problem and conducts experiment with few possible test cases with complexity (6M)	Student solves the problem with few test cases with complexity (4M)	Student fails to solve the problem (2M)
3	Post Experiment Analysis [Viva, Inference] 5M	Demonstrates the simulation/ findings /Hardware results Infers and answer all the Questions posed by Instructor (5M)	Demonstrates results and inference; Able to answer Few Questions posed by Instructor (4M)	Demonstrates results and inference; Unable to answer the Questions posed by Instructor. (3M)	Demonstrates Partial results and inference; Unable to answer the Questions posed by Instructor (2M)	Failed to Demonstrate results and inference; Unable to answer the Questions posed by Instructor (1M)

S. No	Parameter	Descriptors and Score				
		Outstanding	Good	Fair	Poor	Very Poor
4	Report Writing 5M	Report meets all requirements and it is prepared in original and creative way to engage readers (5M)	Report with well-organized content, visuals, graphics, citations and references (4M)	Report is complete and adequate with poor grammar. (3M)	Report is complete, poor grammar and inadequate and failed to organize thoughts (2M)	Report is incomplete, unclear, poor grammar and inadequate (1M)
5	Conduct (Ethics, Safety, Team Work) 5M	Excellent team spirit, strictly follows ethics and safety precautions with good team work (5M)	Follows the safety precautions, practices ethics and poor team work (4M)	Follows safety precautions and ethical practices and failed to exhibit teamwork. (3M)	Followed minimum safety precautions and ethical practices and failed to exhibit team work. (2M)	Does not Follows safety precautions and ethical practices and failed to exhibit team work. (1M)
TOTAL SCORE						

ASSESSMENT PROCEDURE AND AWARD OF SEE

MARKS

S. No.	Parameter	Descriptors and Score				
		Outstanding	Good	Fair	Poor	Very Poor
1	Record 5 M	The content of all the experiments is well-organized, recorded the tables and neatly drawn graphs. Results and discussions are well presented. (5M)	The content of most of the experiments are well-organized, recorded the tables, neatly drawn graphs. Results and discussion are presented. (4M)	The content of the most of experiments are well-organized, recorded the tables, neatly drawn graphs. Results and discussion are not presented (3M)	The content of the few experiments is not well-organized, recorded the tables, neatly drawn graphs. Results and discussion are not presented (2M)	The content of the all the experiments is not well-organized, recorded the tables, neatly drawn graphs. Results and discussion are not presented (1M)
2	Write up about the experiment 10M	Presentation of the given experiment/program is very well organized with the required content, specifications, plan/procedure of the conduct and all the required additional information. (10M)	Presentation of the given experiment/program is organized with the required content, specifications, plan/procedure of the conduct. (8M)	Presentation of the given experiment/program is organized and is without clear specifications and plan/design. (6M)	Presentation of the given experiment/program is minimal with clear specifications and plan/design. (4M)	Presentation of the given experiment/program is minimum without clear specifications and plan/design. (2M)
3	Conduction of the experiment and observations 15M	Conducts experiment / Simulate the problem with proper connections / all possible test cases. All the possible observations are noted. (15M)	Conducts experiment / Simulate the problem with connections / with most of the test cases. Failed to note all observations (12M)	Conducts experiment / Simulate the problem with proper connections / less test cases. (9M)	Conducts experiment / Simulate the problem with improper connections / less test cases. (6M)	Failed to Conducts experiment / Simulate the problem with improper connections / less test cases. (3M)
4	Results & Analysis 10M	Demonstrates the experimental results with adequate analysis / simulation/ findings / obtained results /plotting the graphs. (10M)	Demonstrates the experimental results with required analysis / simulation/ findings / obtained results /plotting the graphs. (8M)	Demonstrates the experimental results, failed to maximum analysis / simulation/ findings. (6M)	Demonstrates the partial experimental results with least analysis / simulation/ findings. (4M)	Failed to Demonstrate the results and least analysis / simulation/ findings. (2M)

S. No.	Parameter	Descriptors and Score				
		Outstanding	Good	Fair	Poor	Very Poor
5	Viva-Voce 10M	Answers most of the questions with good analytical explanation. (10M)	Answers most of the questions with good explanation. (8M)	Answers only few of the questions with good explanation. (6M)	Answers only few of the questions with nominal explanation. (4M)	Failed to answer questions. (2M)
TOTAL SCORE						

GENERAL INSTRUCTIONS FOR LABORATORY CLASSES

DO'S

1. Without Prior permission do not enter into the Laboratory.
2. While entering into the LAB students should wear their ID cards.
3. The Students should come with proper uniform.
4. Students should sign in the LOGIN REGISTER before entering into the laboratory.
5. Students should come with observation and record note book to the laboratory.
6. Students should maintain silence inside the laboratory.
7. After completing the laboratory exercise, make sure to shutdown the system properly

DONT'S

1. Students bringing the bags inside the laboratory..
2. Students wearing slippers/shoes inside the laboratory.
3. Students using the computers in an improper way.
4. Students scribbling on the desk and mishandling the chairs.
5. Students using mobile phones inside the laboratory.
6. Students making noise inside the laboratory.

EXPERIMENT / PRACTICAL -1

Understanding Blockchain Foundations: Elements of Distributed Computing, Elements of Cryptography, Digital Signature

OBJECTIVE OF THE EXPERIMENT:

The objective of the lab is to provide students with a comprehensive grasp of the essential components that serve as the bedrock of blockchain technology. By delving into the realms of Distributed Computing, Cryptography, and Digital Signatures, students will acquire a profound understanding of the fundamental principles that drive the security, transparency, and decentralization inherent in blockchain systems.

OUTCOME OF THE EXPERIMENT:

Distributed Computing:

Understand the principles of distributed computing and its significance in creating decentralized blockchain networks.

Recognize the challenges associated with centralized systems and the advantages of distributed architectures.

Comprehend the role of peer-to-peer networks, nodes, and consensus mechanisms in maintaining a secure and synchronized blockchain ledger.

Grasp the importance of fault tolerance, scalability, and data redundancy in distributed systems.

Cryptography:

Gain a clear understanding of the foundational principles of cryptography and its pivotal role in securing digital communications.

Demonstrate knowledge of core cryptographic operations including encryption, decryption, and hashing.

Explain how cryptographic hash functions contribute to data integrity and authenticity within a blockchain context.

Understand the concept of public-key cryptography and its application in ensuring secure transactions and identity management.

Digital Signatures:

Explain the concept of digital signatures and their significance in verifying the authenticity and integrity of digital documents.

Discuss the importance of digital signatures in achieving non-repudiation within blockchain transactions.

Identify the components involved in creating digital signatures, including private keys, public keys, and the signing process.

Understand the role of digital signatures in verifying the origin and validity of transactions stored on a blockchain.

SYSTEM REQUIREMENTS:

Browsing websites:

1. <https://www.coursera.org/learn/blockchain-basics>
2. <https://blockchainlab.com/>
3. <https://www.web3labs.com/>
4. <https://www.coursera.org/learn/blockchain-foundations-and-use-cases>
5. <https://www.geeksforgeeks.org/what-is-distributed-computing/>

6. <https://www.geeksforgeeks.org/digital-signatures-certificates/>

ALGORITHM / PROCEDURE:

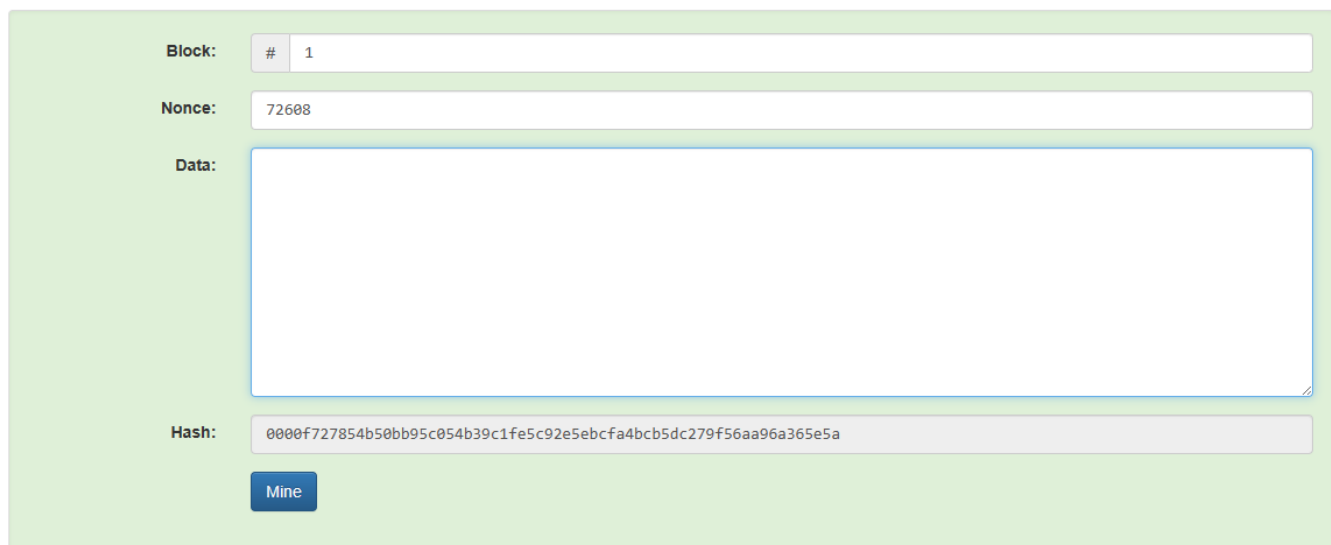
Not required

PSEUDOCODE:

Not required

RESULTS:

Understanding about Blockchain Foundations, Distributed computing and Digital signatures.



The image shows a web-based simulation for mining a blockchain block. It features a light green background. On the left, there are labels for 'Block:', 'Nonce:', 'Data:', and 'Hash:'. The 'Block:' field has a dropdown menu showing '# 1'. The 'Nonce:' field contains the value '72608'. The 'Data:' field is a large, empty rectangular box. The 'Hash:' field displays a long hexadecimal string: '0000f727854b50bb95c054b39c1fe5c92e5ebcfa4bcb5dc279f56aa96a365e5a'. Below the 'Hash:' field is a blue button labeled 'Mine'.

PRE LAB QUESTIONS:

1. What is distributed computing, and how does it differ from centralized computing?
2. What are the advantages of a decentralized network in terms of fault tolerance and scalability?
3. How do nodes communicate in a peer-to-peer network, and why is consensus important in a distributed system?
4. Provide an example of a real-world application where distributed computing is crucial.
5. Define cryptography and its significance in securing digital communications.
6. Explain the differences between encryption and decryption.
7. What is a cryptographic hash function, and how does it contribute to data integrity in blockchain systems?
8. Describe the concept of public-key cryptography and its role in securing transactions.
9. Digital Signature:
10. What is a digital signature, and how does it verify the authenticity and integrity of digital documents?
11. How does a digital signature contribute to non-repudiation in blockchain transactions?
12. Briefly explain the components involved in creating a digital signature.

SAMPLE VIVA VOCE QUESTIONS:

1. Explain the concept of distributed computing and how it relates to the decentralization of blockchain networks.
2. What challenges do centralized systems face that distributed systems aim to overcome?
3. How does consensus play a crucial role in maintaining a consistent blockchain ledger across a distributed network?
4. Can you provide an example of a real-world application that benefits from fault tolerance in a distributed system?
5. Define cryptography and briefly discuss its importance in blockchain technology.
6. How does a cryptographic hash function contribute to ensuring data integrity in a blockchain?
7. Explain the difference between symmetric and asymmetric cryptography. Which one is commonly used in blockchain, and why?
8. Provide an example scenario where public-key cryptography enhances the security of digital transactions.
9. What is a digital signature, and how does it serve as a mechanism for verifying authenticity and integrity?
10. How does a digital signature contribute to the concept of non-repudiation in blockchain transactions?
11. Describe the basic steps involved in creating and verifying a digital signature.
12. Can you think of an application beyond blockchain where digital signatures are used to ensure authenticity?

EXPERIMENT / PRACTICAL -2

Getting familiar with the Ethereum platform and Ethereum virtual machine.

OBJECTIVE OF THE EXPERIMENT:

The objective of the experiment is to provide students with a hands-on introduction to the Ethereum platform and the Ethereum Virtual Machine (EVM). Through this experiential learning, students will gain a solid understanding of Ethereum's foundational components, enabling them to navigate the ecosystem, interact with smart contracts, and comprehend the execution environment of the EVM.

OUTCOME OF THE EXPERIMENT:

Upon completing the experiment, students should be able to:

1. Understand the fundamental principles of the Ethereum platform and its significance in the realm of decentralized applications.
2. Describe the Ethereum Virtual Machine (EVM) and its role in executing smart contracts.
3. Interact with smart contracts on the Ethereum network using Ethereum addresses and transactions.
4. Write and deploy a simple smart contract using the Solidity programming language.
5. Explain the concept of blockchain immutability and transparency through the deployment of smart contracts.

SYSTEM REQUIREMENTS:

Browsing website <https://ethereum.org/en/developers/docs/evm/>

Computer with internet access.

Modern web browser for exploring Ethereum-related websites and tools.

ALGORITHM / PROCEDURE:

Introduction to Ethereum:

Research and read introductory materials about Ethereum to understand its basic concepts, architecture, and use cases.

Exploring Ethereum Ecosystem:

Visit the official Ethereum website (ethereum.org) to explore information about the platform, its features, and latest updates.

Learn about Ether (ETH), the native cryptocurrency of the Ethereum platform.

Understanding Smart Contracts:

Learn about smart contracts, their definition, and their significance in the context of Ethereum.

Understand how smart contracts enable the execution of decentralized applications.

Introduction to Ethereum Virtual Machine (EVM):

Research and read about the Ethereum Virtual Machine (EVM) and its role in executing smart contracts.

Consensus Mechanism:

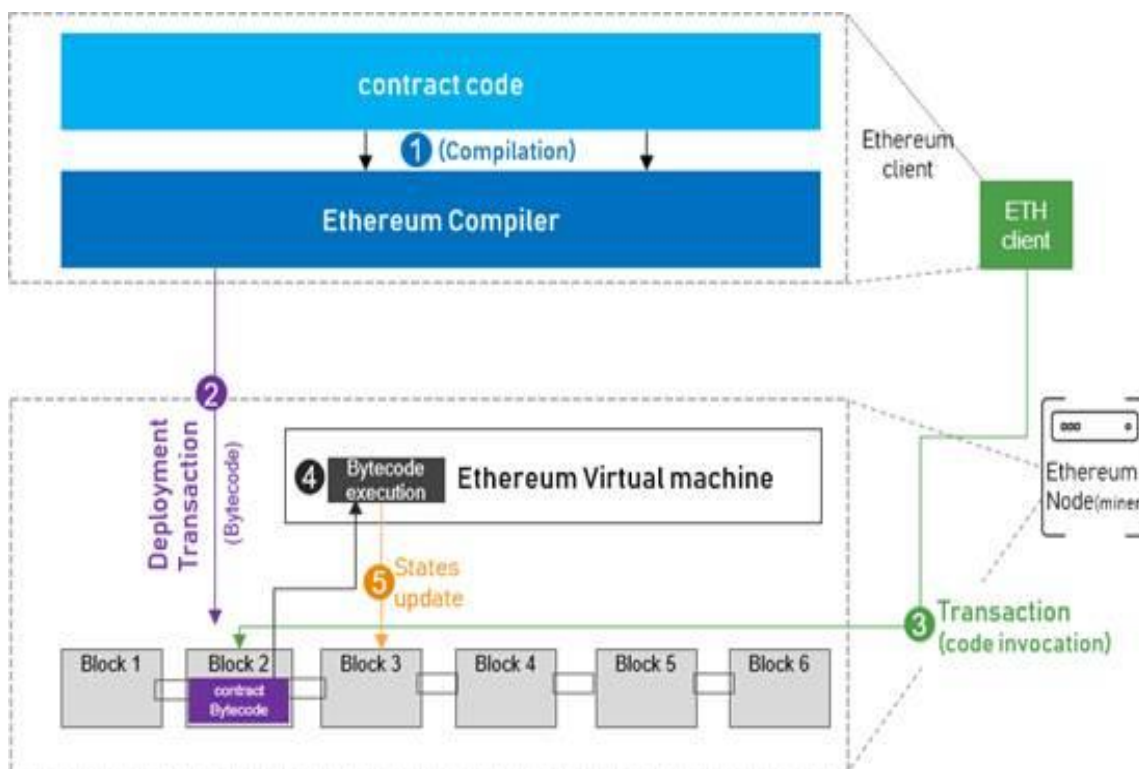
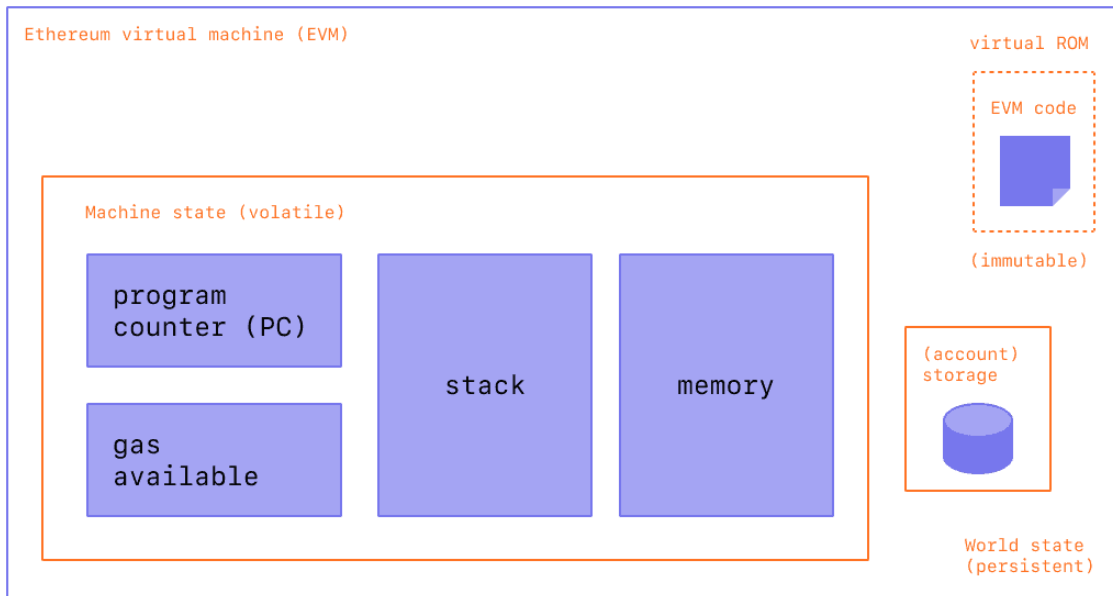
Learn about Ethereum's consensus mechanism, which is based on Proof of Stake (PoS) in Ethereum 2.0.

PSEUDOCODE:

N/A (This experiment involves theoretical learning and exploration, without coding or pseudocode implementation.)

RESULTS:

Participants have gained familiarity with the Ethereum platform and Ethereum Virtual Machine (EVM). They understand the basics of Ethereum's architecture, the role of smart contracts, and how the EVM facilitates the execution of these contracts in a decentralized manner.



PRE LAB QUESTIONS:

1. What is Ethereum, and how does it differ from Bitcoin in terms of its primary functionality?
2. How does Ethereum achieve decentralization, and what role do nodes play in maintaining the network?
3. What is a smart contract, and how does it contribute to the functionality of the Ethereum platform?
4. Ethereum Virtual Machine (EVM) and Smart Contracts:
5. What is the Ethereum Virtual Machine (EVM), and why is it a crucial component of the Ethereum ecosystem?
6. How does the EVM ensure deterministic execution of code across the network?

7. Briefly explain the concept of gas in the context of Ethereum transactions and smart contract execution.
8. What is an Ethereum address, and how is it used to identify students on the network?
9. How can students interact with deployed smart contracts using Ethereum addresses and transactions?
10. Provide an example scenario where interacting with a smart contract could be beneficial.

SAMPLE VIVA VOCE QUESTIONS:

1. What is the Solidity programming language, and what role does it play in creating smart contracts?
2. Describe the steps involved in deploying a smart contract onto the Ethereum network.
3. How does the immutability of the blockchain contribute to the security and transparency of deployed smart contracts?
4. Benefits and Future Considerations:
5. Why is it important to gain hands-on experience with the Ethereum platform and its virtual machine?
6. How might understanding Ethereum's components benefit professionals in the fields of technology, finance, or research?
7. Can you envision potential challenges or opportunities that may arise from the widespread adoption of Ethereum technology in various industries?
8. These prelab questions are designed to assess students' familiarity with Ethereum basics, the Ethereum Virtual Machine (EVM), smart contracts, and their potential applications. Students' responses will provide insights into their existing knowledge and serve as a foundation for discussions during the experiment.

EXPERIMENT / PRACTICAL -3

Introduction to solidity program structure, compilation, and deployment environment.

OBJECTIVE OF THE EXPERIMENT:

The objective of this experiment is to introduce students to the fundamental concepts of the Solidity programming language, its program structure, the process of compilation, and the deployment environment. By the end of the experiment, students will have a foundational understanding of Solidity, enabling them to write, compile, and deploy simple smart contracts on a test Ethereum network.

OUTCOME OF THE EXPERIMENT:

Compilation and Bytecode:

Students will understand the steps involved in compiling Solidity code into bytecode, ready for execution on the Ethereum Virtual Machine (EVM).

They will be able to explain the significance of the Application Binary Interface (ABI) in facilitating interactions with smart contracts.

Development Environment Setup:

Students will be proficient in setting up a development environment for writing, testing, and

deploying Solidity smart contracts.

They will have practical experience using tools like Remix or Truffle for managing contract compilation and deployment.

Writing and Deploying Smart Contracts:

Students will demonstrate the ability to write a basic smart contract using Solidity's syntax and features.

SYSTEM REQUIREMENTS:

Browsing website <https://soliditylang.org>

Computer with internet access.

Solidity compiler (available online or integrated into development tools).

Ethereum development environment (e.g., Remix, Truffle, Ganache).

ALGORITHM / PROCEDURE:

Solidity Program Structure:

Study the basic structure of a Solidity smart contract, including contract declaration, state variables, functions, and modifiers.

Write a simple Solidity smart contract with state variables and basic functions.

Compilation of Solidity Code:

Use an online Solidity compiler (e.g., Remix) or an integrated development environment (e.g., Truffle) to compile the written Solidity code.

Observe the compilation process and review any error messages or warnings.

Deployment Environment:

Set up a local development blockchain network using tools like Ganache.

Deploy the compiled smart contract to the local development network.

Explore Ethereum testnets (e.g., Ropsten) and obtain test Ether for deployment

PSEUDOCODE:

```
contract SimpleContract {  
    uint256 public data;  
  
    function setData(uint256 _data) public {  
        data = _data;  
    }  
  
    function getData() public view returns (uint256) {  
        return data;  
    }  
}
```

RESULTS:

Participants have gained an understanding of the Solidity program structure, the process of compiling Solidity code, and the deployment environment for smart contracts. They have successfully written a simple Solidity smart contract, compiled it using a Solidity compiler or development tool, and deployed it to a local development network or testnet.

PRE LAB QUESTIONS:

1. What is Solidity, and why is it a crucial programming language in the context of blockchain and smart contract development?
2. Explain the concept of a smart contract and its significance in the Ethereum platform.
3. What are variables, functions, and control structures in programming, and how do they relate to Solidity?
4. Briefly describe the process of compiling code in programming languages. How is it relevant to Solidity?
5. What is bytecode, and how does it relate to the Ethereum Virtual Machine (EVM)?
6. Why is the Application Binary Interface (ABI) important when interacting with smart contracts on the Ethereum network?
7. What tools or software can be used to write and deploy Solidity smart contracts?
8. Why is it essential to have a development environment set up before engaging in smart contract development?
9. How does a test Ethereum network differ from the main Ethereum network, and why would you use it for deploying and testing contracts?

SAMPLE VIVA VOCE QUESTIONS:

1. What role does the Ethereum Virtual Machine (EVM) play in executing smart contracts?
2. Describe the significance of Ethereum addresses when it comes to deploying and interacting with smart contracts.
3. How does the immutability of the blockchain impact the deployment and execution of smart contracts?
4. What benefits do you anticipate gaining from the hands-on experience with Solidity, compilation, and deployment covered in this experiment?
5. How might understanding Solidity, compilation, and deployment be valuable for professionals in fields such as software development, finance, or research?
6. Can you identify potential challenges or complexities that individuals might face when working with smart contracts and deploying them on the Ethereum network?

EXPERIMENT / PRACTICAL -4

Creating and deploying the simple smart contract to store and get "Hello World", on the Blockchain network.

OBJECTIVE OF THE EXPERIMENT:

The objective of this experiment is to provide students with a hands-on experience in creating, compiling, and deploying a basic smart contract onto a blockchain network. Specifically, students will write a smart contract using Solidity that stores the string "Hello World" on the blockchain and provides a function to retrieve it. Through this experiment, students will gain practical insights into the process of contract creation, deployment, and interaction on a real blockchain network.

OUTCOME OF THE EXPERIMENT:

Smart Contract Creation:

Students will have written a simple smart contract using the Solidity programming language. They will have defined a state variable to store the string "Hello World" within the contract.

Compilation and Bytecode Generation:

Students will understand the process of transforming Solidity code into bytecode suitable for execution by the Ethereum Virtual Machine (EVM).

They will grasp the concept of bytecode and its role in enabling contract execution on the blockchain.

Deployment on the Blockchain Network:

Students will have successfully deployed their created smart contract onto a real blockchain network.

They will have gained practical experience with transaction initiation and witnessing deployment transactions being added to the blockchain.

Interacting with the Contract:

Students will have used tools like Remix or blockchain explorers to interact with the deployed smart contract.

They will have invoked the contract's function to retrieve the stored "Hello World" string from the blockchain.

General Learning Outcomes:

By the end of the experiment, students will have developed broader skills and insights, including:

A practical understanding of the contract development lifecycle, from creation to deployment and interaction.

A tangible demonstration of how blockchain immutability and transparency apply to storing and retrieving data.

Familiarity with the process of writing, compiling, and deploying smart contracts, setting the stage for more complex contract development.

SYSTEM REQUIREMENTS:

Remix IDE

ALGORITHM/PROCEDURE:

Step 1: Create a new Solidity file

Create a new file with a .sol extension, such as HelloWorld.sol, and open it in a Solidity development environment or text editor.

Step 2: Write the smart contract code Inside the .sol file, write the code for the smart contract. In this case, we'll create a contract called HelloWorld with a state variable to store the string and a getter function to retrieve it.

```
pragma solidity ^0.8.0;
```

```
contract HelloWorld {  
    string public message;  
  
    function setMessage(string memory _message) public {  
        message = _message;  
    }  
  
    function getMessage() public view returns (string memory) {  
        return message;  
    }  
}
```

In this contract, the message variable is of type string and is publicly accessible. The setMessage function allows you to set the message, and the getMessage function retrieves the message.

Step 3: Compile the smart contract

Use a Solidity compiler or development tool (such as Remix IDE, Truffle, or Hardhat) to compile the smart contract code. Make sure to select the desired Solidity compiler version specified in the pragma directive of your contract.

Step 4: Deploy the smart contract

Choose a blockchain network or Ethereum-compatible network (e.g., Ganache, Ropsten, or the Ethereum mainnet) to deploy your contract. Use a deployment tool or platform to configure the deployment parameters and deploy the compiled bytecode and ABI of your contract.

Step 5: Interact with the smart contract

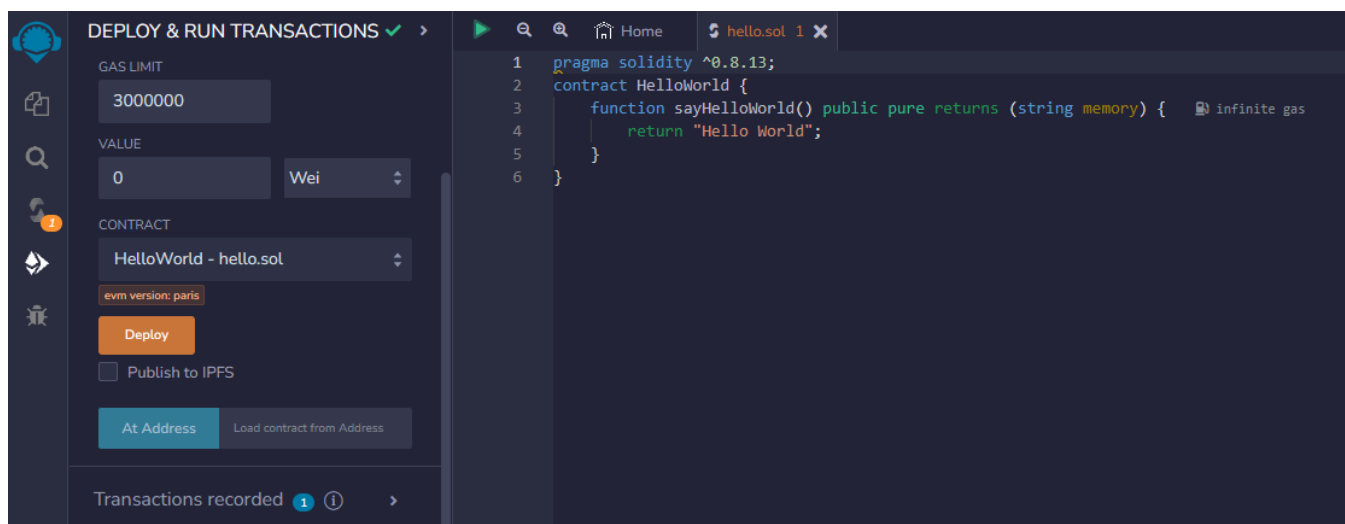
Once deployed, you can interact with the contract using its functions. For example, you can call the setMessage function to set a new message, and then use the getMessage function to retrieve the updated message.

PSEUDOCODE:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.0 <0.9.0;
```

```
contract SimpleStorage {
    uint storedData; // State variable
    // ...
}
```

RESULTS:



PRE LAB QUESTIONS:

1. What is a smart contract in the context of blockchain technology, and how does it differ from traditional contracts?
2. Have you heard of the Solidity programming language before? If yes, briefly describe its role in creating smart contracts.

3. What is a state variable in a smart contract, and why is it important for storing data on the blockchain?
4. Explain the concept of blockchain immutability and why data stored on a blockchain cannot be easily altered.
5. How can users interact with smart contracts on a blockchain network, and what types of operations can be performed on them?
6. Why is it advantageous to deploy smart contracts on a test network before deploying them on the main Ethereum network?
7. Describe the process of deploying a smart contract onto a blockchain network. What role do transactions play in this process?
8. How does the Ethereum Virtual Machine (EVM) ensure consistent execution of smart contract code across the network?
9. In terms of Solidity code, how can you define a function that allows users to retrieve a stored value from a deployed smart contract?

SAMPLE VIVA VOCE QUESTIONS:

1. What benefits do you expect to gain from this experiment in terms of understanding smart contract development and blockchain technology?
2. How might the ability to write, deploy, and interact with smart contracts be valuable for professionals in fields like software development, finance, or supply chain management?
3. Can you envision applications beyond storing "Hello World" where the principles of smart contracts and blockchain transparency could be useful?

EXPERIMENT / PRACTICAL -5

Develop a smart contract to create a function setter and getter to set and get a value.

OBJECTIVE OF THE EXPERIMENT:

The objective of this experiment is to guide students through the process of designing, coding, compiling, and deploying a smart contract that incorporates both setter and getter functions. Students will develop a basic smart contract using the Solidity programming language that allows users to set and retrieve a value. By the end of the experiment, students will have gained practical experience in writing contract functions and interacting with a deployed contract on a blockchain network.

OUTCOME OF THE EXPERIMENT:

Smart Contract Design and Implementation:

Students will have successfully designed a smart contract that incorporates both setter and getter functions for value management.

They will have implemented the Solidity code necessary to define these functions within the contract.

Setter and Getter Functions:

Students will comprehend the purpose and functionality of the setter function in allowing users to input and store a value within the contract.

They will have implemented the getter function to enable the retrieval of the stored value from the contract.

Compilation and Bytecode Generation:

Students will have gained hands-on experience compiling the Solidity code into bytecode, preparing the contract for deployment on the Ethereum Virtual Machine (EVM).

They will understand the significance of bytecode generation in enabling contract execution on a blockchain network.

Deployment and Interaction:

Students will have successfully deployed their developed smart contract onto a real blockchain network.

They will have interacted with the deployed contract by invoking the setter and getter functions to set and retrieve values.

SYSTEM REQUIREMENTS:

Remix IDE

ALGORITHM / PROCEDURE:

Step 1: Create a new Solidity file

Create a new file with a .sol extension, such as HelloWorld.sol, and open it in a Solidity development environment or text editor.

Step 2: Write the smart contract code Inside the .sol file, write the code for the smart contract. In this case, we'll create a contract called HelloWorld with a state variable to store the string and a getter function to retrieve it.

pragma solidity ^0.8.0;

```
pragma solidity ^0.8.0;

contract ValueContract {
    uint private value; // Private variable to store the value

    // Setter function to set the value
    function setValue(uint newValue) public {
        value = newValue;
    }

    // Getter function to retrieve the value
    function getValue() public view returns (uint) {
        return value;
    }
}
```

In this contract, the message variable is of type string and is publicly accessible. The setMessage function allows you to set the message, and the getMessage function retrieves the message.

Step 3: Compile the smart contract

Use a Solidity compiler or development tool (such as Remix IDE, Truffle, or Hardhat) to compile the smart contract code. Make sure to select the desired Solidity compiler version specified in the pragma directive of your contract.

Step 4: Deploy the smart contract

Choose a blockchain network or Ethereum-compatible network (e.g., Ganache, Ropsten, or the Ethereum mainnet) to deploy your contract. Use a deployment tool or platform to configure the deployment parameters and deploy the compiled bytecode and ABI of your contract.

Step 5: Interact with the smart contract

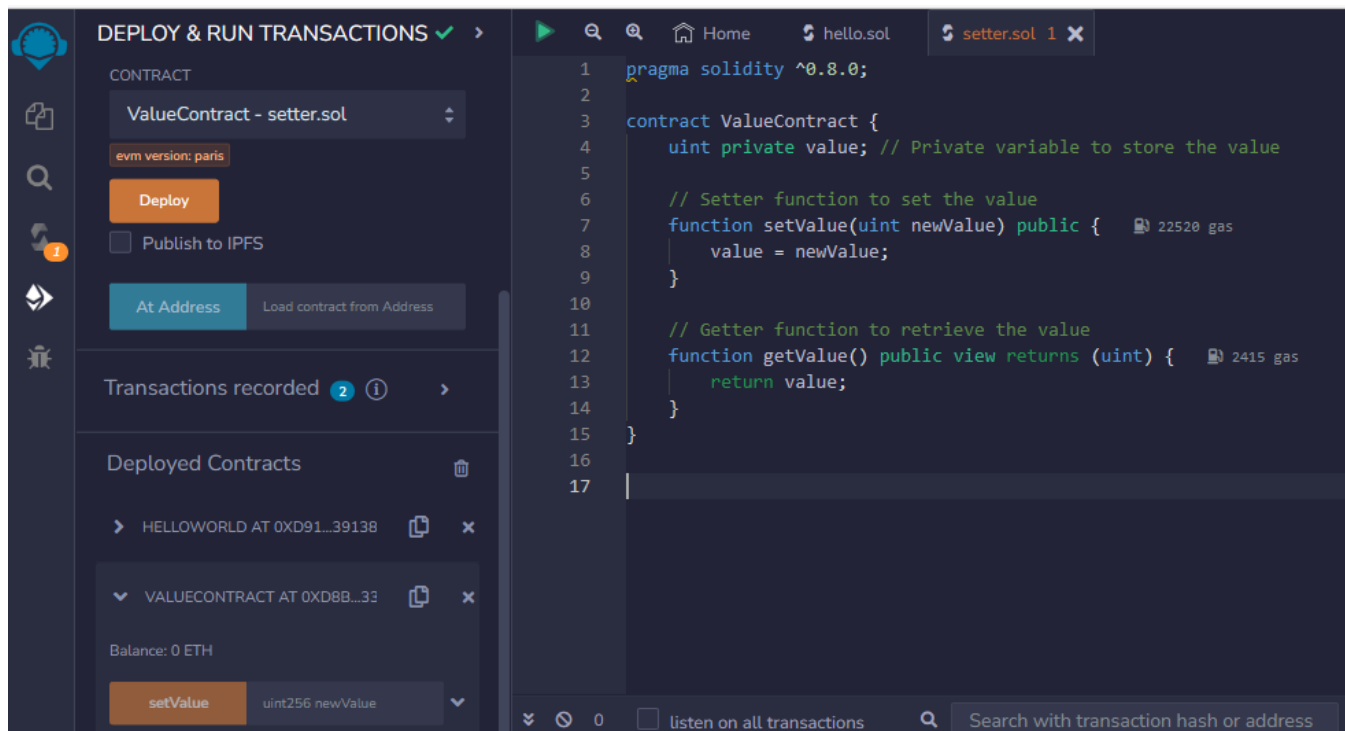
Once deployed, you can interact with the contract using its functions. For example, you can call the setMessage function to set a new message, and then use the getMessage function to retrieve the updated message.

PSEUDOCODE:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.0 <0.9.0;
```

```
contract SimpleStorage {
    uint storedData; // State variable
    // ...
}
```

RESULTS:



PRE LAB QUESTIONS:

1. What is a smart contract, and how does it differ from a traditional legal contract?
2. How do smart contracts contribute to the transparency, security, and automation of processes within blockchain technology?
3. Briefly explain the significance of Solidity in the development of smart contracts.
4. In the context of a smart contract, what is the purpose of a setter function? Provide an example scenario where a setter function would be useful.
5. Describe the functionality of a getter function in a smart contract. How does it enable users to access data?
6. What is the significance of having both setter and getter functions in a single smart contract?

7. What are state variables in a Solidity smart contract, and why are they important?
8. How would you define a state variable in Solidity that can hold an integer value?
9. Explain how you would write Solidity code to implement a setter function that allows users to set a value in a smart contract.
10. Describe the steps involved in compiling a Solidity smart contract and generating the bytecode for deployment.

SAMPLE VIVA VOCE QUESTIONS:

1. What is the role of the Ethereum Virtual Machine (EVM) in executing functions within a deployed smart contract?
2. How can users interact with a deployed smart contract on a blockchain network, and what are the primary mechanisms for invoking functions?
3. Why is the ability to create and deploy smart contracts with setter and getter functions important for various industries beyond the technology sector?
4. Can you anticipate any challenges that students might face during the process of writing, deploying, and interacting with a smart contract?
5. How might a deeper understanding of smart contract development influence decision-making in fields such as finance, supply chain management, or healthcare?

EXPERIMENT / PRACTICAL -6

Develop Solidity contracts to illustrate inheritance and polymorphism.

OBJECTIVE OF THE EXPERIMENT:

The objective of this experiment is to provide students with a hands-on opportunity to explore and understand the concepts of inheritance and polymorphism in the context of Solidity smart contract development. Students will create Solidity contracts that demonstrate the inheritance relationship between smart contracts and showcase the principles of polymorphism. Through this experiment, students will enhance their understanding of object-oriented programming concepts within the Solidity programming language.

OUTCOME OF THE EXPERIMENT:

Understanding Inheritance:

Students will comprehend the concept of inheritance and its application in smart contract development.

They will recognize the benefits of creating reusable base contracts that can be extended by derived contracts.

Implementing Inheritance:

Students will have successfully implemented multiple Solidity contracts that showcase the inheritance relationship.

They will understand how derived contracts can inherit properties and behaviors from parent contracts.

Exploring Polymorphism:

Students will grasp the principle of polymorphism and its manifestation in the Solidity language.

They will have created derived contracts with varying behaviors, demonstrating polymorphic behavior when interacting with common interfaces.

Advanced Solidity Skills:

Students will have gained advanced proficiency in Solidity syntax and best practices for designing contracts with inheritance and polymorphism.

They will understand how to structure contracts to achieve modularity, reusability, and maintainability.

SYSTEM REQUIREMENTS:

Remix IDE

ALGORITHM / PROCEDURE:

Step 1: Create a new Solidity file

Create a new file with a .sol extension, such as HelloWorld.sol, and open it in a Solidity development environment or text editor.

Step 2: Write the smart contract code Inside the .sol file, write the code for the smart contract. In this case, we'll create a contract called HelloWorld with a state variable to store the string and a getter function to retrieve it.

```
pragma solidity ^0.8.0;
```

```
pragma solidity ^0.8.0;
```

```
// Parent contract
```

```
contract Shape {  
    string internal shapeType;
```

```
    constructor(string memory _type) {  
        shapeType = _type;  
    }
```

```
    function getType() public view returns (string memory) {  
        return shapeType;  
    }
```

```
}
```

```
// Child contract inheriting from Shape
```

```
contract Rectangle is Shape {  
    uint private width;  
    uint private height;
```

```
    constructor(uint _width, uint _height) Shape("Rectangle") {  
        width = _width;  
        height = _height;  
    }
```

```
    function getArea() public view returns (uint) {  
        return width * height;  
    }
```

```
}
```

```
// Another child contract inheriting from Shape
```

```
contract Circle is Shape {  
    uint private radius;
```

```

constructor(uint _radius) Shape("Circle") {
    radius = _radius;
}

function getArea() public view returns (uint) {
    return calculateArea();
}

function calculateArea() private view returns (uint) {
    uint pi = 3; // Simplified value of pi for demonstration purposes
    return (pi * radius * radius) / 4;
}
}

```

6b. polymorphism

```

pragma solidity ^0.8.0;

// Parent contract
abstract contract Shape {
    string internal shapeType;

    constructor(string memory _type) {
        shapeType = _type;
    }

    function getArea() public virtual view returns (uint);
}

// Child contract inheriting from Shape
contract Rectangle is Shape {
    uint private width;
    uint private height;

    constructor(uint _width, uint _height) Shape("Rectangle") {
        width = _width;
        height = _height;
    }

    function getArea() public view virtual override returns (uint) {
        return width * height;
    }
}

// Another child contract inheriting from Shape
contract Circle is Shape {
    uint private radius;
    uint private constant pi = 31415; // Using fixed-point arithmetic

    constructor(uint _radius) Shape("Circle") {
        radius = _radius;
    }
}

```



```

    }

    function getArea() public view virtual override returns (uint) {
        return (pi * radius * radius) / 10000; // Adjusting the scale for fixed-point arithmetic
    }
}

// Contract to interact with shapes
contract ShapeInteraction {
    function calculateArea(Shape shape) public view returns (uint) {
        return shape.getArea();
    }
}

```

Step 3: Compile the smart contract

Use a Solidity compiler or development tool (such as Remix IDE, Truffle, or Hardhat) to compile the smart contract code. Make sure to select the desired Solidity compiler version specified in the pragma directive of your contract.

Step 4: Deploy the smart contract

Choose a blockchain network or Ethereum-compatible network (e.g., Ganache, Ropsten, or the Ethereum mainnet) to deploy your contract. Use a deployment tool or platform to configure the deployment parameters and deploy the compiled bytecode and ABI of your contract.

Step 5: Interact with the smart contract

Once deployed, you can interact with the contract using its functions. For example, you can call the setMessage function to set a new message, and then use the getMessage function to retrieve the updated message.

PSEUDOCODE:

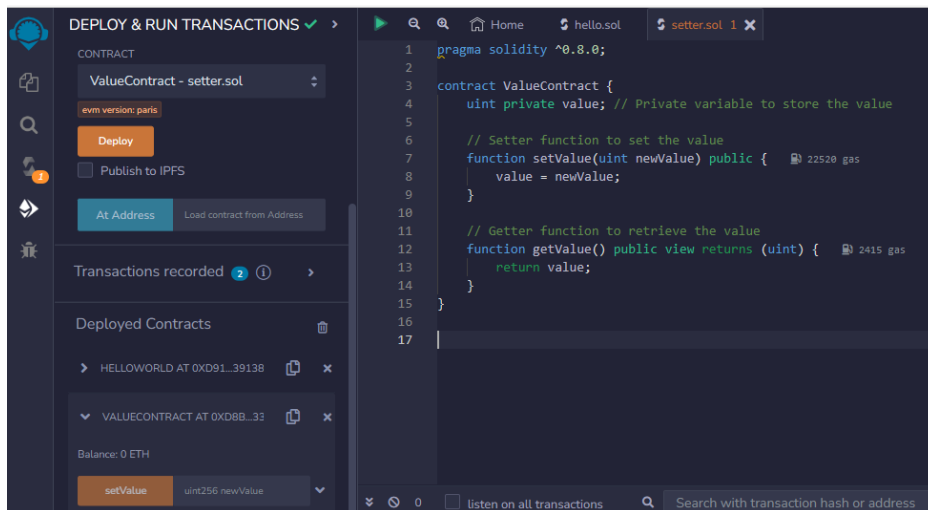
```

// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.0 <0.9.0;

contract SimpleStorage {
    uint storedData; // State variable
    // ...
}

```

RESULTS:



PRE LAB QUESTIONS:

1. What is the concept of inheritance in the context of programming, and how does it relate to object-oriented programming?
2. How can inheritance enhance code reusability and modularity in software development?
3. Can you provide an example scenario where the concept of inheritance could be applied to smart contract development?
4. Define the principle of polymorphism in programming. How does it allow different objects to be treated uniformly through a common interface?
5. What are the benefits of using polymorphism in software design and development?
6. How might polymorphism be useful when creating multiple versions of a smart contract with varying behaviors?

SAMPLE VIVA VOCE QUESTIONS:

1. Briefly explain the role of Solidity in blockchain technology and smart contract development.
2. What is the purpose of interfaces in Solidity contracts, and how do they relate to polymorphism?
3. Can you differentiate between a base contract and a derived contract in the context of Solidity?
4. Why is it beneficial to design contracts in a modular and reusable manner?
5. How can a base contract be designed to allow for the extension and specialization of derived contracts?
6. In the context of Solidity, what are some best practices for ensuring clean and maintainable contract code?

EXPERIMENT / PRACTICAL -7

Familiarize with the working of Remix Ethereum tool.

OBJECTIVE OF THE EXPERIMENT:

The objective of this experiment is to provide students with a hands-on introduction to the Remix Ethereum development tool. Students will become familiar with the features and functionalities of Remix, an integrated development environment (IDE) for writing, testing, and deploying Ethereum smart contracts. By the end of the experiment, students should be comfortable using Remix for various stages of smart contract development, from writing Solidity code to deploying contracts on a blockchain network.

OUTCOME OF THE EXPERIMENT:

Introduction to Remix:

Students will be familiar with the purpose and role of the Remix Ethereum tool as an integrated development environment (IDE) for Ethereum smart contract development.

They will have navigated Remix's user interface, exploring different panels and tabs to understand its layout and features.

Solidity Contract Interaction:

Students will have successfully created, written, and edited Solidity smart contracts using the Remix editor.

They will understand the process of writing contract code within Remix's development environment.

Compilation and Deployment:

Students will have compiled Solidity code within Remix, observing the compilation process and gaining insights into the generated bytecode.

They will have deployed a smart contract onto a test Ethereum network using Remix's deployment tools, witnessing the deployment process.

Testing and Interaction:

Students will have utilized Remix's testing features to write and execute unit tests for their smart contracts, demonstrating proficiency in contract testing.

They will have interacted with deployed contracts through Remix's interface, invoking functions and observing the resulting outputs.

SYSTEM REQUIREMENTS:

Remix IDE

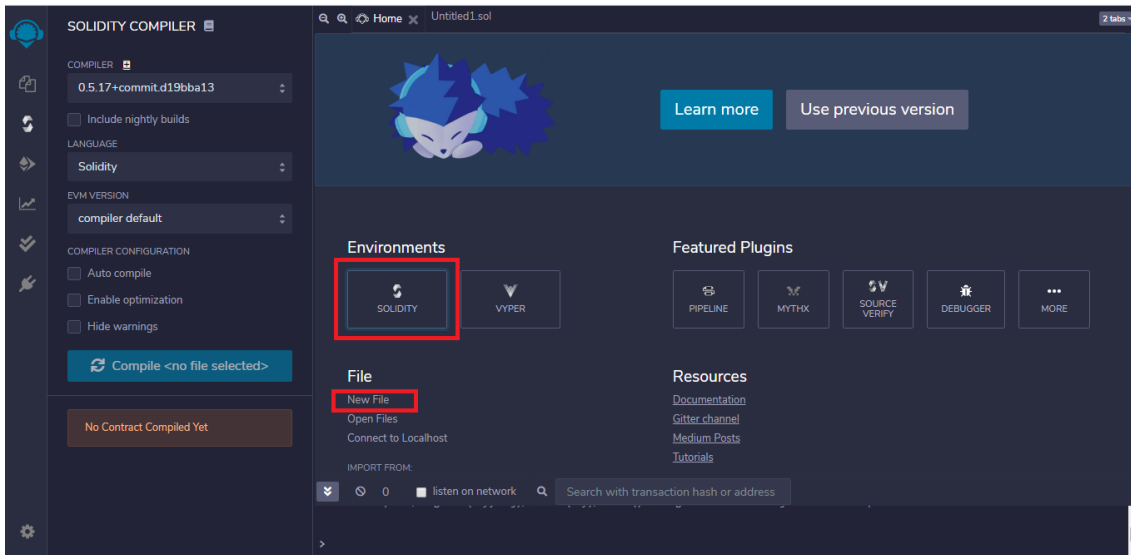
Computer with internet access.

Web browser (Google Chrome, Mozilla Firefox) for accessing Remix Ethereum Tool.

ALGORITHM / PROCEDURE:

Remix IDE is generally used to compile and run Solidity smart contracts. Below are the steps for the compilation, execution, and debugging of the smart contract.

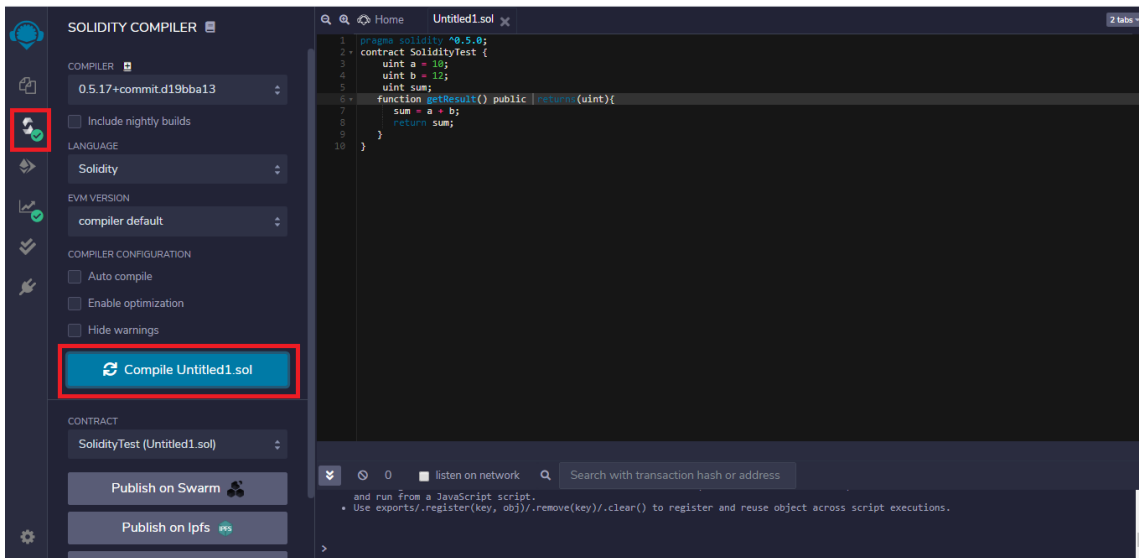
Step 1: Open Remix IDE on any of your browsers, select on New File and click on Solidity to choose the environment.



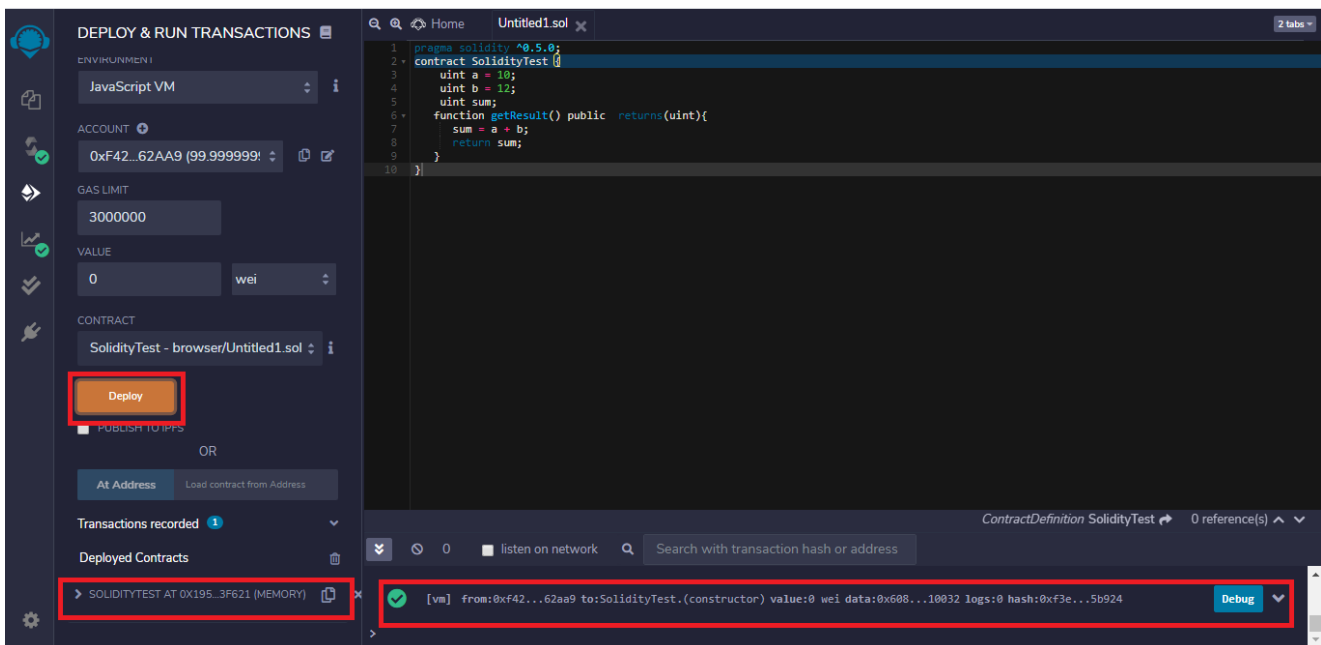
Step 2: Write the Smart contract in the code section, and click the Compile button under the Compiler window to compile the contract.

Solidity

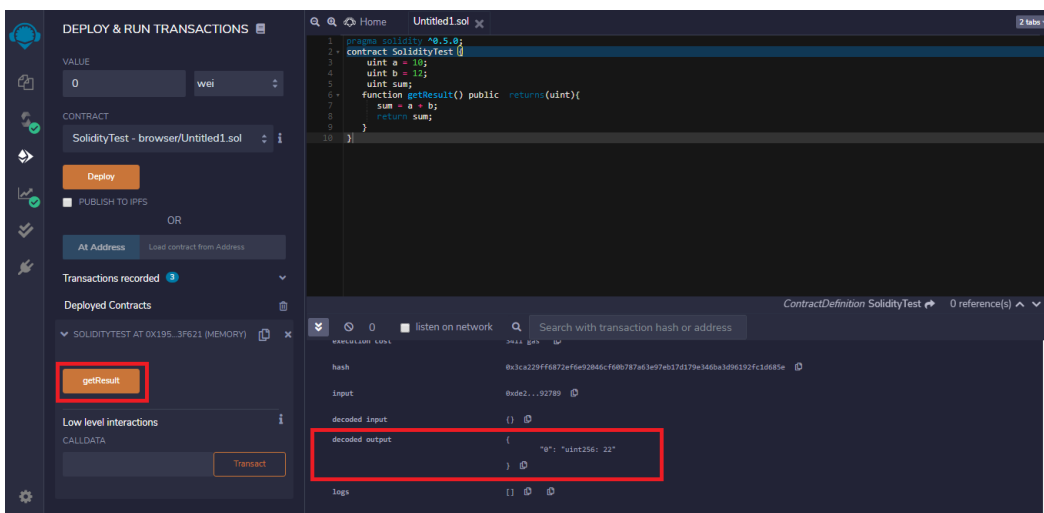
```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.5.0;
contract SolidityTest{
    uint a=10;
    uint b=12;
    uint sum;
    function getResult() public returns(uint){
        sum=a+b;
        return sum;
    }
}
```



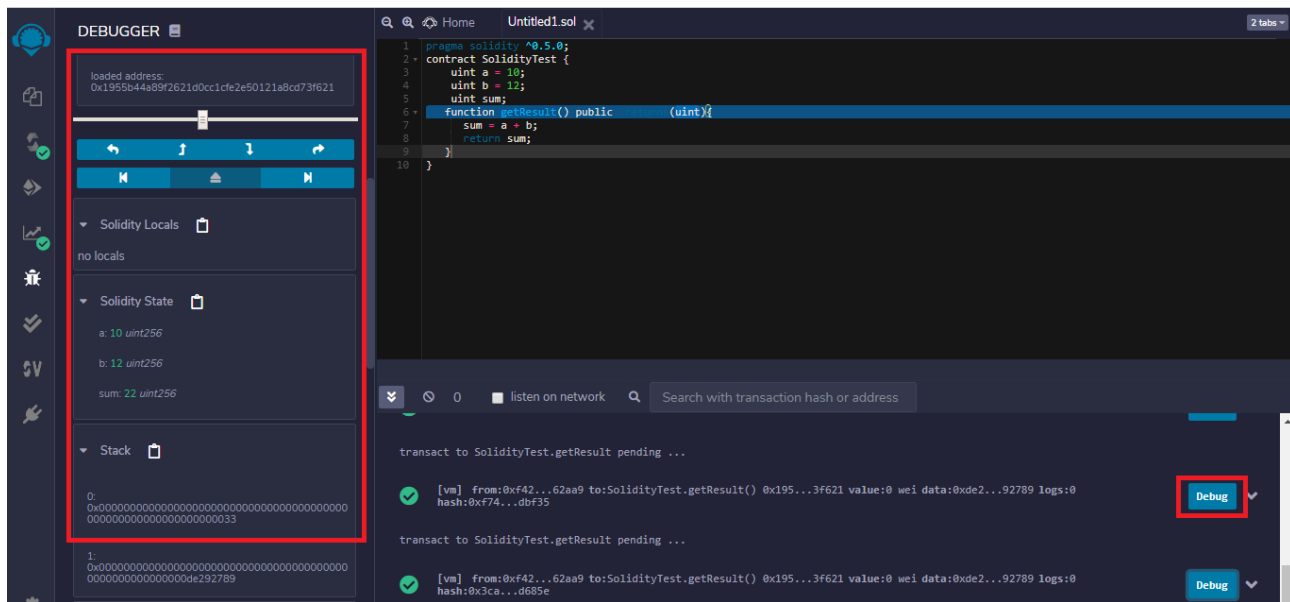
Step 3: To execute the code, click on the Deploy button under Deploy and Run Transactions window.



Step 4: After deploying the code click on the method calls under the drop-down of deployed contracts to run the program, and for output, check to click on the drop-down on the console.



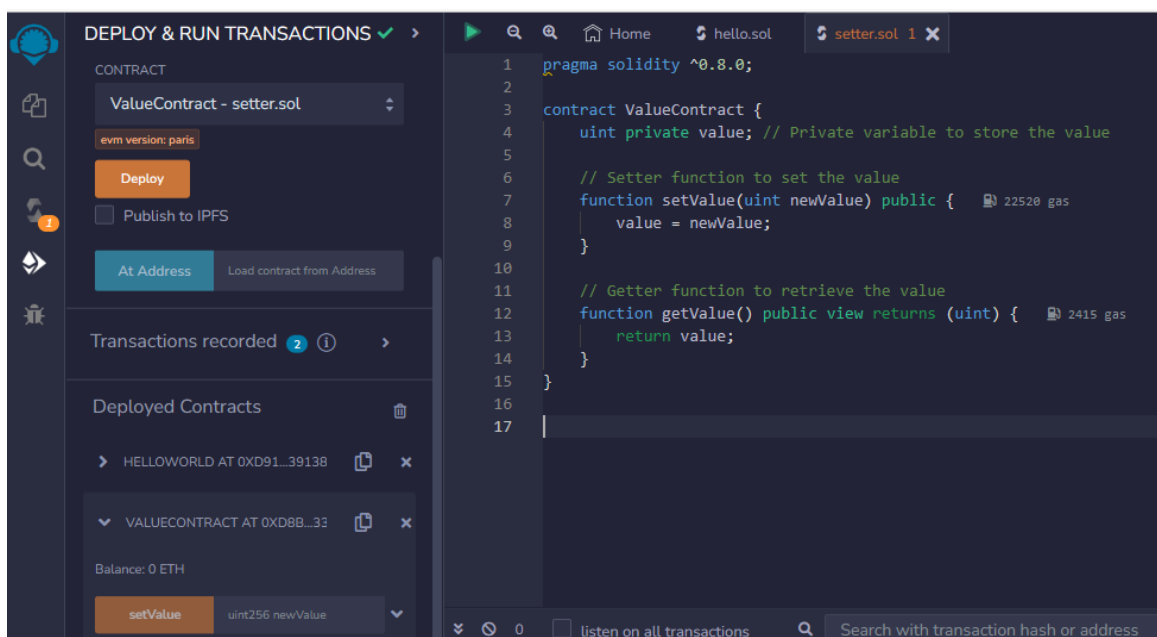
Step 5: For debugging click on the Debug button corresponding to the method call in the console. Here you can check each function call and variable assignments.



PSEUDOCODE:

N/A (This experiment involves using the user interface of the Remix Ethereum Tool without writing specific code or pseudocode.)

RESULTS:



PRE LAB QUESTIONS:

1. What is the Remix Ethereum tool, and what is its primary purpose in the context of blockchain development?
2. How does Remix facilitate the process of writing, compiling, deploying, and testing Ethereum smart contracts?
3. Can you briefly describe the user interface of Remix? What are some of the key panels or tabs that you might encounter?

4. What is Solidity, and why is it an essential programming language for creating Ethereum smart contracts?
5. How might you create a new Solidity smart contract within the Remix Ethereum tool? What are the steps involved?
6. In the context of Remix, what is the significance of the editor panel, and how does it assist in writing Solidity code?

SAMPLE VIVA VOCE QUESTIONS:

1. What is the purpose of compiling Solidity code, and why is it necessary before deploying a smart contract?
2. What is bytecode in the context of Ethereum smart contracts, and how does Remix assist in generating bytecode from compiled code?
3. Explain how Remix's deployment tools help users deploy smart contracts onto an Ethereum network. What information is required for deployment?
4. How can you use Remix's testing features to ensure the functionality and correctness of your smart contracts?
5. What are some common tasks you might perform when interacting with a deployed smart contract through Remix's interface?
6. Why might it be beneficial to deploy and test contracts on a test Ethereum network before deploying them on the main Ethereum network?

EXPERIMENT / PRACTICAL -8

Design a smart contract on Remix Ethereum to print the array of integers and its length.

OBJECTIVE OF THE EXPERIMENT:

The objective of this experiment is to guide students through the process of designing, coding, and deploying a smart contract using the Remix Ethereum development tool. Students will create a smart contract that contains an array of integers and includes functions to print the array elements and the length of the array. Through this experiment, students will gain hands-on experience in working with arrays within smart contracts, as well as using Remix to interact with and deploy contracts on the Ethereum network.

OUTCOME OF THE EXPERIMENT:

Smart Contract Design and Implementation:

Students will have successfully designed a smart contract that includes an array of integers.

They will have implemented the Solidity code necessary to define the array and functions for printing array elements and its length.

Array Manipulation and Output:

Students will understand how to manipulate and access array elements within a Solidity smart contract.

They will have gained proficiency in writing functions that allow the contract to print the array's elements and its length.

Compilation and Bytecode Generation:

Students will have observed the process of compiling Solidity code within the Remix Ethereum tool.

They will understand the role of bytecode generation in preparing the contract for deployment on the Ethereum network.

Deployment and Interaction:

Students will have successfully deployed their developed smart contract onto a test Ethereum network.

They will have interacted with the deployed contract through Remix's interface, invoking functions to print array data.

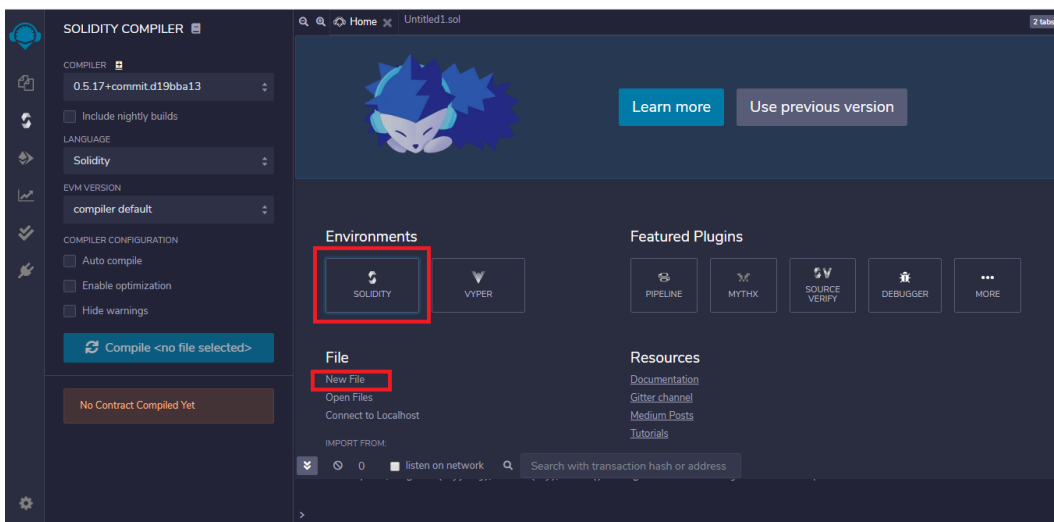
SYSTEM REQUIREMENTS:

Remix IDE

ALGORITHM/ PROCEDURE:

Remix IDE is generally used to compile and run Solidity smart contracts. Below are the steps for the compilation, execution, and debugging of the smart contract.

Step 1: Open Remix IDE on any of your browsers, select on New File and click on Solidity to choose the environment.



Step 2: Write the Smart contract in the code section, and click the Compile button under the Compiler window to compile the contract.

Solidity

```
pragma solidity ^0.8.0;

contract ArrayPrinter {
    uint[] public integers;

    constructor() {
        integers = [1, 2, 3, 4, 5];
    }

    function getArray() public view returns (uint[] memory) {
        return integers;
    }
}
```



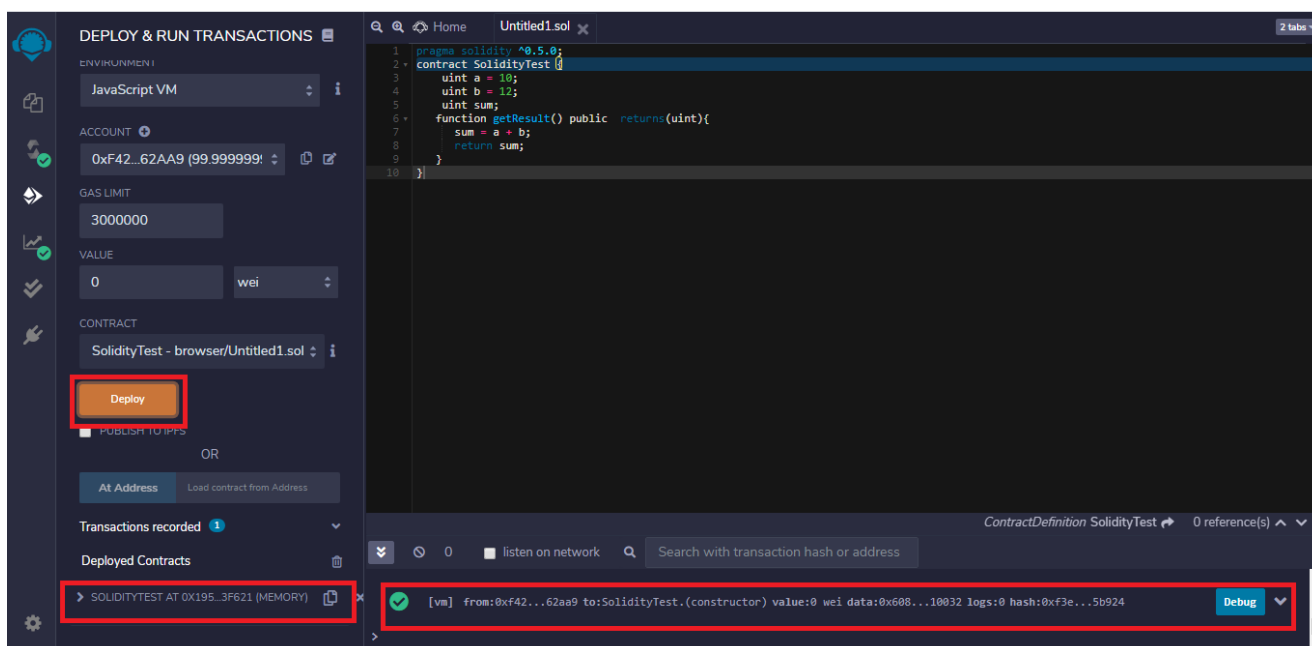
```

}

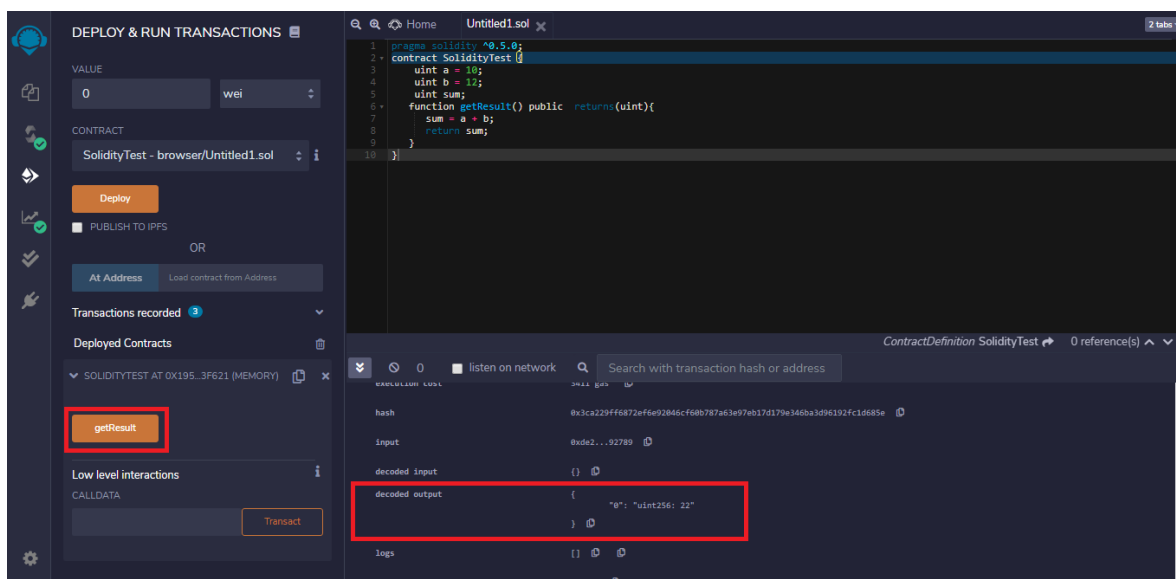
function getArrayLength() public view returns (uint) {
    return integers.length;
}
}

```

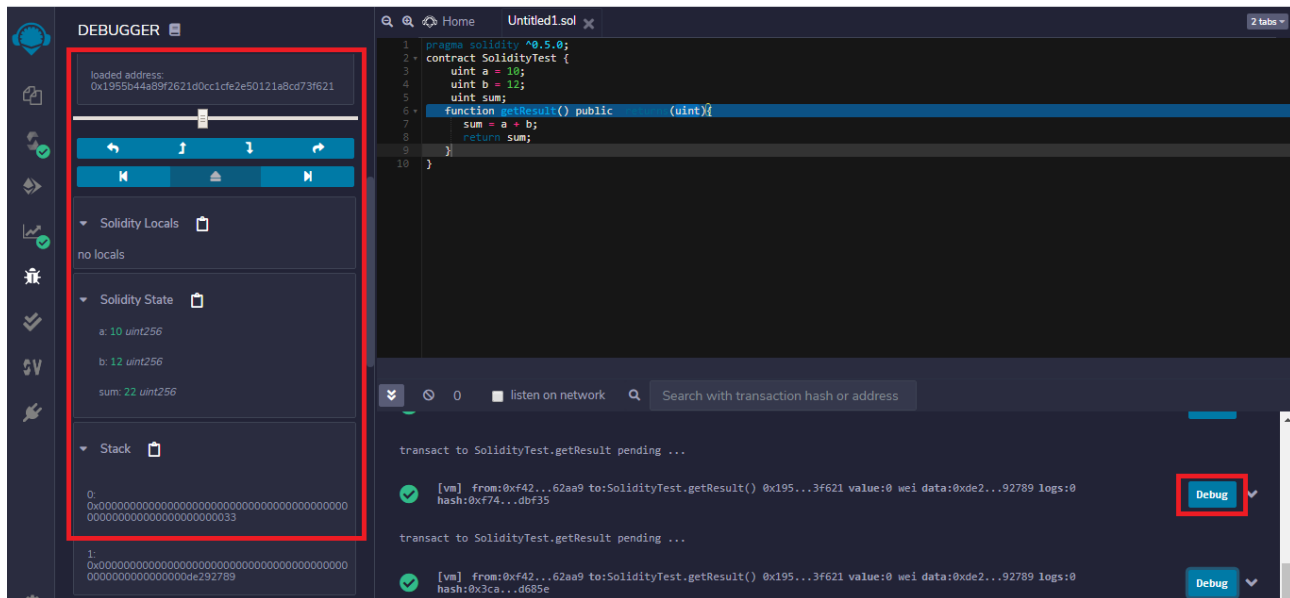
Step 3: To execute the code, click on the Deploy button under Deploy and Run Transactions window.



Step 4: After deploying the code click on the method calls under the drop-down of deployed contracts to run the program, and for output, check to click on the drop-down on the console.



Step 5: For debugging click on the Debug button corresponding to the method call in the console. Here you can check each function call and variable assignments.

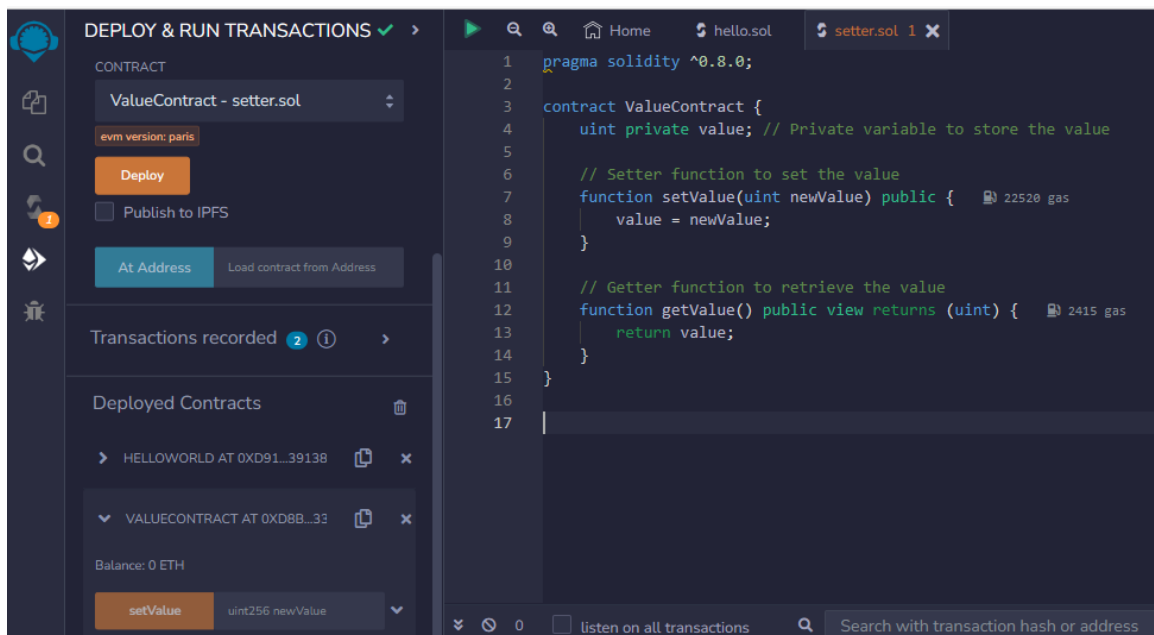


PSEUDOCODE:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.0 <0.9.0;
```

```
contract SimpleStorage {
    uint storedData; // State variable
    // ...
}
```

RESULTS:



PRE LAB QUESTIONS:

1. What is the purpose of a smart contract in the context of blockchain technology?
2. How do smart contracts contribute to transparency and automation of processes in

various industries?

3. Can you provide an example scenario where a smart contract with an array manipulation feature would be valuable?
4. In programming, what is an array, and how is it used to store and manage a collection of elements?
5. How can arrays be beneficial in the context of blockchain and smart contract development?
6. Describe a real-world use case where an array of data would be useful in a blockchain-based application.
7. Briefly explain the role of Solidity in creating smart contracts for the Ethereum network.

SAMPLE VIVA VOCE QUESTIONS:

1. What is the Remix Ethereum tool, and how does it assist in smart contract development?
2. How might you create a new Solidity smart contract using Remix's interface?
3. In the context of smart contracts, what is the role of a function?
4. How would you approach designing a smart contract that includes an array of integers and functions to print the array elements and length?
5. What is the significance of interacting with deployed contracts through Remix's interface?

EXPERIMENT / PRACTICAL -9

Setup a Simple Ethereum wallet and use it to send and receive Ethers.

OBJECTIVE OF THE EXPERIMENT:

The main objective of this experiment is to guide students through the process of setting up a simple Ethereum wallet and using it to perform basic transactions such as sending and receiving Ether (ETH). Students will become familiar with the practical aspects of cryptocurrency management, exploring the fundamental features of Ethereum wallets and gaining hands-on experience in conducting transactions on the Ethereum network.

OUTCOME OF THE EXPERIMENT:

Ethereum Wallet Setup:

Students will have successfully created a simple Ethereum wallet using a user-friendly wallet provider or software.

They will understand the key components of a wallet, including the public address and private key.

Transaction Initiation:

Students will have initiated a transaction to send Ether to another Ethereum address, gaining experience in transaction setup.

They will understand the concept of gas and gas fees as well as the process of confirming transactions.

Transaction Reception:

Students will have received Ether by sharing their wallet's public address and observing the transaction on the Ethereum blockchain.

They will appreciate the transparency and immutability of blockchain-based

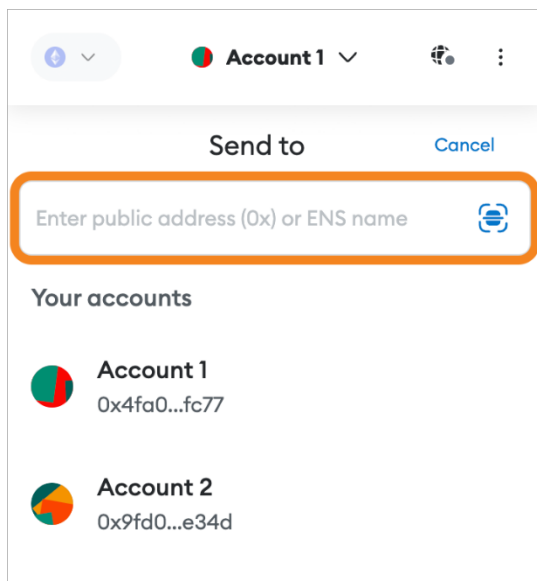
SYSTEM REQUIREMENTS:

Metamask Wallet

ALGORITHM/ PROCEDURE:

From the landing page of your wallet, make sure you're in the account from which you want to transact, and hit the 'Send' button in the middle of the screen.

Now you need to input the public address of the recipient. If you already have addresses saved in your address book, they will appear now.



Enter the amount of tokens you want to send and click next.

Account 1

Send

Account 2
0x9fd0d0040b5a67d768015c764caa659555ace34d

Asset: ETH
Balance: 0.00163471 ETH

Amount: .001 ETH
\$1.87 USD

Gas (estimated) 0.00028596 ETH
Likely in < 30 seconds Max fee: 0.00038531 ETH

Cancel Next

Now you're presented with the estimated gas fees of your transaction, which you can also adjust. Double-checking the recipient address before clicking 'Confirm' to proceed with the transaction is generally a good idea.

< Edit Ethereum Mainnet

Account 1 → Account 2

SENDING ETH

0.001 ETH
\$1.87

Market >

Gas (estimated) \$0.58 0.00031055 ETH
Likely in < 30 seconds Max fee: 0.00041851 ETH

Total \$2.45 0.00131055 ETH
Amount + gas fee Max amount: 0.00141851 ETH

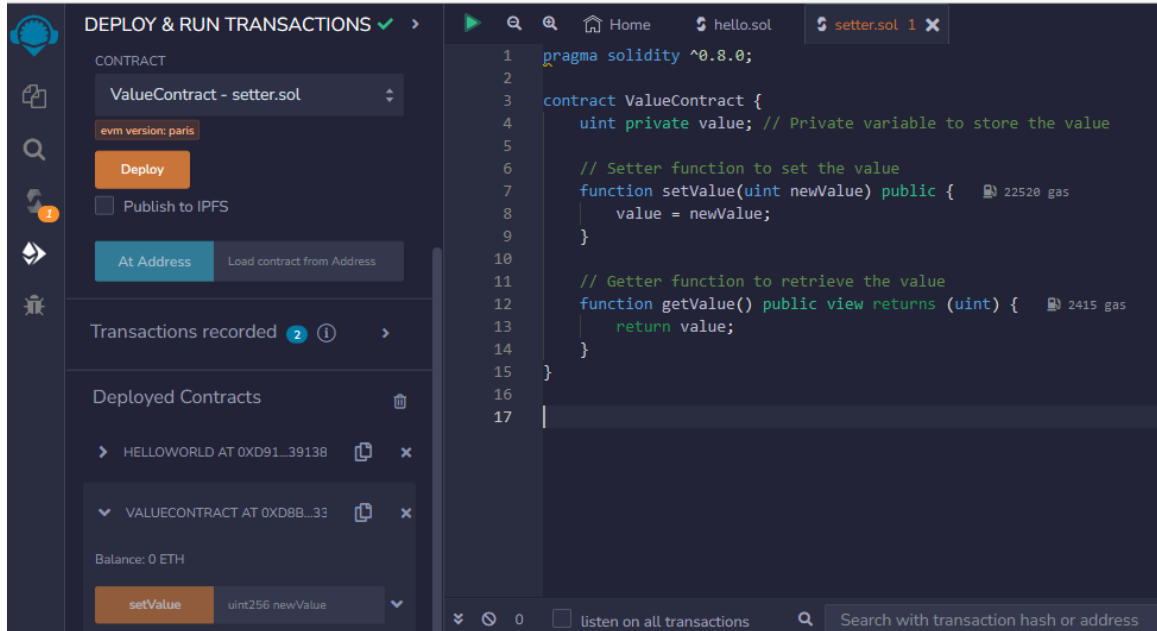
Reject Confirm

You will then be redirected to the homepage, where you can see a list of your recent transactions on the 'Activity' tab.

PSEUDOCODE:

N/A (This experiment involves using the user interface of Metamask wallet Tool without writing specific code or pseudocode.)

RESULTS:



PRE LAB QUESTIONS:

1. What is a cryptocurrency, and how does it differ from traditional fiat currencies?
2. Can you briefly explain the concept of a blockchain and its role in enabling secure and transparent cryptocurrency transactions?
3. How do public and private keys relate to cryptocurrency transactions and wallet security?
4. What is the Ethereum network, and how does it stand out from other blockchain networks?
5. Why is a wallet necessary when dealing with cryptocurrencies like Ether (ETH) on the Ethereum network?
6. How does an Ethereum wallet store and provide access to your cryptocurrencies?
7. In the context of cryptocurrency transactions, what is meant by "sending" and "receiving" cryptocurrency?

SAMPLE VIVA VOCE QUESTIONS:

1. Can you explain the concept of gas in Ethereum transactions and its importance in the network?
2. What are gas fees, and why do they vary based on network activity?
3. Why is it crucial to securely store your wallet's private key and backup phrase?
4. What are some security practices you should follow to prevent unauthorized access to your Ethereum wallet?
5. Describe a scenario where verifying the recipient's address is essential before initiating a cryptocurrency transaction.

EXPERIMENT / PRACTICAL -10

Hyperledger Fabric Demo.

OBJECTIVE OF THE EXPERIMENT:

The objective of this experiment is to provide students with a hands-on demonstration of Hyperledger Fabric, an enterprise-grade permissioned blockchain framework. Students will gain exposure to the key features and functionalities of Hyperledger Fabric, allowing them to understand how it can be utilized for creating private, scalable, and secure blockchain networks for various business applications.

OUTCOME OF THE EXPERIMENT:

Understanding Hyperledger Fabric:

Students will be familiar with the fundamental concepts of Hyperledger Fabric as an enterprise-grade permissioned blockchain framework.

They will grasp the differences between Hyperledger Fabric and public blockchain networks in terms of design, permission models, and use cases.

Network Setup and Configuration:

Students will have witnessed the process of setting up and configuring a basic Hyperledger Fabric network.

They will comprehend the roles of various network components, including organizations, peers, and ordering nodes.

Smart Contracts Deployment:

Students will have observed the deployment of smart contracts (chaincode) on the Hyperledger Fabric network.

They will understand the steps involved in writing, packaging, installing, and invoking chaincode.

Transaction Processing and Consensus:

Students will have gained insight into the transaction lifecycle in Hyperledger Fabric, from proposal to consensus.

They will understand the role of endorsement policies, validation, and consensus mechanisms.

SYSTEM REQUIREMENTS:

Computer with internet access.

Terminal or command-line interface for executing Hyperledger Fabric commands.

Docker and Docker Compose installed on the system.

ALGORITHM / PROCEDURE:

Navigate to Hyperledger Fabric Samples:

Open your terminal and navigate to the directory where you have cloned the Hyperledger Fabric Samples repository.

Start the Fabric Network: Navigate to the "first-network" directory:

bashCopy code

```
cd first-network
```

Run the script to bring up the network:

bashCopy code

```
./byfn.sh generate ./byfn.sh up
```

Create and Deploy Chaincode: In a new terminal window, navigate to the "chaincode" directory:

Invoke and Query Chaincode: After instantiation, invoke and query the chaincode using the CLI container:

PSEUDOCODE:

N/A (This experiment involves using the command-line interface to interact with Hyperledger Fabric, without writing specific code or pseudocode.)

RESULTS:

Participants have successfully witnessed a demonstration of the Hyperledger Fabric blockchain framework. They gained an understanding of key concepts, such as organizations, peers, ordering nodes, and channels, and experienced the process of setting up a basic Hyperledger Fabric network, deploying chaincode, and interacting with the network.



PRE LAB QUESTIONS:

1. What is Hyperledger Fabric, and what distinguishes it from public blockchain networks like Ethereum?
2. Briefly explain the concept of a permissioned blockchain framework and how it aligns with enterprise requirements.
3. Why might enterprises choose to use Hyperledger Fabric for building private blockchain networks?
4. What are some key components of a Hyperledger Fabric network, and what roles do they play?

5. Describe the purpose of an ordering node and a peer in a Hyperledger Fabric network.
6. How are channels utilized in Hyperledger Fabric to enhance privacy and data segregation?
7. What is the purpose of smart contracts (chaincode) in Hyperledger Fabric, and how do they differ from traditional contracts?

SAMPLE VIVA VOCE QUESTIONS:

1. In the context of Hyperledger Fabric, what is the process of deploying and invoking chaincode?
2. How does Hyperledger Fabric ensure endorsement and validation of transactions involving chaincode?
3. Explain the sequence of steps involved in processing a transaction within a Hyperledger Fabric network.
4. How is consensus achieved in a Hyperledger Fabric network, and why is it essential?
5. Why is it important for network students to endorse transactions before they are committed to the ledger?

Experiment Beyond curriculum:

EXPERIMENT / PRACTICAL -1

Develop a multi-signature wallet smart contract. This contract would require multiple parties to approve a transaction before it can be executed.

OBJECTIVE OF THE EXPERIMENT:

The objective of this experiment is to guide students in developing a multi-signature wallet smart contract. The smart contract will enforce a requirement for multiple parties to approve a transaction before it can be executed on the blockchain.

OUTCOME OF THE EXPERIMENT:

Smart Contract Development:

Proficiency in writing a multi-signature wallet smart contract with the required logic for transaction approval.

Understanding of how to define and manage multiple signatories for the approval process.

Blockchain Interaction:

Ability to interact with the deployed multi-signature wallet smart contract on the blockchain.

Knowledge of submitting transactions and observing the contract's behavior during the multi-signature approval process.

SYSTEM REQUIREMENTS:

Development environment for the chosen blockchain platform (e.g., Ethereum, Binance Smart

Chain).

Suitable Integrated Development Environment (IDE) or text editor.

MetaMask or similar wallet extension for testing (if applicable).

ALGORITHM / PROCEDURE:

Smart Contract Creation:

Start by defining the required number of signatures for transaction approval.

Write a smart contract that stores signatories' addresses and tracks their approvals.

Implement logic to enforce the multi-signature requirement for transaction execution.

Deployment:

Deploy the smart contract to the chosen blockchain network using the appropriate deployment tool (e.g., Remix, Truffle).

Interaction:

Use a wallet extension (e.g., MetaMask) to interact with the deployed contract.

Submit a transaction and observe the process of obtaining approvals from multiple parties.

Confirm that the transaction executes only after the required number of approvals are obtained.

PSEUDOCODE:

```
contract MultiSignatureWallet {
    address[] public signatories;
    mapping(address => bool) public approved;
    constructor(address[] memory _signatories)
    { signatories = _signatories; }
    function approve() public { require(isSignatory(msg.sender), "Not a signatory");
    require(!approved[msg.sender], "Already approved"); approved[msg.sender] = true; }

    function isSignatory(address _address) private view returns (bool) { for (uint256 i = 0; i <
    signatories.length; i++) { if (signatories[i] == _address) { return true; } } return false; }
    function executeTransaction() public { require(isApproved(), "Not approved yet"); // Perform
    transaction logic // ... }
    function isApproved() private view returns (bool) { uint256 approvalCount = 0; for (uint256 i =
    0; i < signatories.length; i++) { if (approved[signatories[i]]) { approvalCount++; } } return
    approvalCount >= (signatories.length / 2) + 1; } }
```

RESULTS:

Students successfully developed a multi-signature wallet smart contract that enforces transaction approval from multiple parties before execution. They interacted with the contract, submitted transactions, and observed the multi-signature approval process in action.

PRE LAB QUESTIONS:

1. What is the purpose of a multi-signature wallet smart contract?
2. Why might organizations or individuals use a multi-signature wallet for transactions?
3. How can a multi-signature wallet smart contract enhance security in blockchain-based transactions?

SAMPLE VIVA VOCE QUESTIONS:

1. Explain the concept of a multi-signature wallet smart contract and its significance.
2. How does the developed smart contract ensure that transactions require approval from

- multiple parties?
3. Describe the process of creating and deploying a multi-signature wallet smart contract.
 4. In what situations might a multi-signature wallet be preferred over a single-signature wallet?
 5. How does the contract determine when a transaction is eligible for execution based on multi-signature approvals?

Experiment Beyond curriculum

EXPERIMENT / PRACTICAL -2

Building a Simple Decentralized Application using Ethereum.

OBJECTIVE OF THE EXPERIMENT:

The main objective of this experiment is to guide students through the process of building a simple decentralized application (DApp) using Ethereum as the backend.

OUTCOME OF THE EXPERIMENT:

Smart Contract Development:

Proficiency in writing a simple smart contract using Solidity.

Understanding of deploying smart contracts to the Ethereum blockchain.

Web3 Integration:

Knowledge of how to integrate Web3.js library for communication between the frontend and the Ethereum blockchain.

Blockchain Interaction:

Ability to interact with the deployed smart contract using a frontend interface.

SYSTEM REQUIREMENTS:

Development environment with Ethereum development tools (Ganache, Remix, Truffle).

Text editor or Integrated Development Environment (IDE).

Modern web browser for testing the DApp.

ALGORITHM / PROCEDURE:

Smart Contract Creation:

Define the data and functionality of the smart contract you want to develop.

Write the smart contract in Solidity, defining its state variables and functions.

Smart Contract Deployment:

Deploy the smart contract using a development blockchain network like Ganache or a testnet like Ropsten.

Note the contract address for interaction.

Frontend Development:

Create a basic HTML and JavaScript frontend interface to interact with the smart contract.

Integrate the Web3.js library to establish communication between the frontend and the blockchain.

Interact with the Smart Contract:

Enable users to interact with the DApp through the frontend interface.

Implement functions to send transactions to the deployed smart contract.

PSEUDOCODE:

```
// Example SimpleStorage smart contract pragma solidity ^0.8.0;
contract SimpleStorage
{ uint256 public data; function setData(uint256 _data) public { data = _data; }
function getData() public view returns (uint256) { return data; } }
```

RESULTS:

Students successfully built a simple decentralized application using Ethereum as the backend. They developed and deployed a basic smart contract, integrated it with a frontend interface using Web3.js, and interacted with the DApp by sending transactions to the smart contract and retrieving data from the Ethereum blockchain.

PRE LAB QUESTIONS:

1. What is a decentralized application (DApp), and why are they gaining popularity?
2. How does Ethereum enable the development of decentralized applications?
3. What is the role of a smart contract in a decentralized application?

SAMPLE VIVA VOCE QUESTIONS:

1. Describe the process of building a decentralized application using Ethereum as the backend.
2. How do smart contracts enhance the functionality and interaction of decentralized applications?
3. Explain the importance of integrating the Web3.js library in a decentralized application.
4. What is the purpose of a smart contract address, and how is it used for interaction?
5. How can users interact with a decentralized application's frontend to send transactions to the Ethereum blockchain?
- 6.

Experiment Beyond curriculum EXPERIMENT / PRACTICAL -3

Design and implement a crowdfunding smart contract where users can contribute Ether to a project, and the funds are released only when a predefined goal is reached.

OBJECTIVE OF THE EXPERIMENT:

The objective of this experiment is to guide students in designing and implementing a crowdfunding smart contract on the Ethereum blockchain.

OUTCOME OF THE EXPERIMENT:

Smart Contract Design: Students will learn how to design a smart contract that enables users to contribute Ether to a project and ensures that funds are released only when a predefined fundraising goal is reached.

Solidity Programming: Students will gain hands-on experience in writing Solidity code for creating the crowdfunding smart contract.

Blockchain Interaction: Students will interact with the deployed smart contract to contribute funds, monitor the progress towards the goal, and trigger fund release.

SYSTEM REQUIREMENTS:

Ethereum development environment (Ganache, Remix, Truffle).

Text editor or Integrated Development Environment (IDE).

Web3.js library for frontend interaction (optional).

ALGORITHM / PROCEDURE:

Smart Contract Design:

Define the data and functionality required for the crowdfunding smart contract, including contributor tracking, goal amount, and fund release conditions.

Smart Contract Implementation:

Write the smart contract in Solidity, incorporating the defined data and functionality.

Smart Contract Deployment:

Deploy the smart contract on a development blockchain network (e.g., Ganache) or a testnet (e.g., Ropsten).

Frontend Interface (Optional):

Create a basic HTML and JavaScript frontend interface to interact with the smart contract.

Integrate the Web3.js library to enable interaction between the frontend and the smart contract.

Interact with the Crowdfunding Smart Contract:

Use the frontend interface (or command-line tools) to contribute Ether to the smart contract.

Monitor the progress towards the fundraising goal.

Trigger the release of funds when the goal is met.

PSEUDOCODE:

```
// Example Crowdfunding smart contract
pragma solidity ^0.8.0;
```

```
contract Crowdfunding {
    address public owner;
    uint256 public goal;
    uint256 public raisedAmount;
    mapping(address => uint256) public contributions;
    bool public goalReached;

    constructor(uint256 _goal) {
        owner = msg.sender;
        goal = _goal;
    }
}
```

RESULTS:

Students successfully designed and implemented a crowdfunding smart contract that allows users to contribute Ether to a project and ensures that funds are released only when a predefined goal is reached. They interacted with the smart contract, contributed funds, and verified the release of funds when the goal was met.

PRE LAB QUESTIONS:

1. What is the purpose of a crowdfunding smart contract in the context of blockchain technology?
2. How does a crowdfunding smart contract differ from traditional crowdfunding platforms?
3. What is the role of the "goalReached" variable in the smart contract?

SAMPLE VIVA VOCE QUESTIONS:

1. Explain the concept of a crowdfunding smart contract and its primary features.
2. How does the developed smart contract ensure that funds are released only when the fundraising goal is reached?
3. Describe the key components of the crowdfunding smart contract's Solidity code.
4. What is the purpose of the "contributions" mapping in the smart contract?
5. How can users interact with the deployed crowdfunding smart contract to contribute funds and monitor the progress towards the goal?

Experiment Beyond curriculum

EXPERIMENT / PRACTICAL -4

Develop a web interface that interacts with your deployed smart contracts using the Web3.js library.

OBJECTIVE OF THE EXPERIMENT:

The main objective of this experiment is to guide students in designing and implementing a crowdfunding smart contract on the Ethereum blockchain.

OUTCOME OF THE EXPERIMENT:

Smart Contract Design:

Proficiency in designing a crowdfunding smart contract with features such as contribution tracking and goal verification.

Solidity Programming:

Understanding of writing Solidity code to define the structure and behavior of a crowdfunding smart contract.

Blockchain Interaction:

Ability to contribute Ether to the smart contract, monitor the status of fundraising, and trigger the release of funds when the goal is reached.

SYSTEM REQUIREMENTS:

Ethereum development environment (Ganache, Remix, Truffle).

Text editor or Integrated Development Environment (IDE).

Web3.js library for frontend interaction (optional).

ALGORITHM / PROCEDURE:

Smart Contract Design:

Define the data and functionality required for the crowdfunding smart contract, including contributor tracking, goal amount, and fund release conditions.

Smart Contract Implementation:

Write the smart contract in Solidity, incorporating the defined data and functionality.

Smart Contract Deployment:

Deploy the smart contract on a development blockchain network (e.g., Ganache) or a testnet (e.g., Ropsten).

Frontend Interface (Optional):

Create a basic HTML and JavaScript frontend interface to interact with the smart contract.

Integrate the Web3.js library to enable interaction between the frontend and the smart contract.

Interact with the Crowdfunding Smart Contract:

Use the frontend interface (or command-line tools) to contribute Ether to the smart contract.

Monitor the progress towards the fundraising goal.

Trigger the release of funds when the goal is met.

PSEUDOCODE:

```
// Example Crowdfunding smart contract pragma solidity ^0.8.0;
contract Crowdfunding { address public owner; uint256 public goal; uint256 public
raisedAmount; mapping(address => uint256) public contributions; bool public goalReached;
constructor(uint256 _goal) { owner = msg.sender; goal = _goal; } function contribute() public
payable { require(!goalReached, "Goal already reached"); contributions[msg.sender] +=
msg.value; raisedAmount += msg.value; if (raisedAmount >= goal) { goalReached = true; } }
function withdrawFunds() public { require(msg.sender == owner, "Only owner can withdraw
funds"); require(goalReached, "Goal not reached yet"); payable(owner).transfer(raisedAmount); raisedAmount = 0; } }
```

RESULTS:

Students successfully designed and implemented a crowdfunding smart contract that allows users to contribute Ether to a project and ensures that funds are released only when a predefined goal is reached. They interacted with the smart contract, contributed funds, and verified the release of funds when the goal was met.

PRE LAB QUESTIONS:

1. What is the purpose of integrating the Web3.js library into a web interface?
2. How does the Web3.js library enable interaction with smart contracts on the Ethereum blockchain?
3. Explain the role of the Web3.js library in interacting with Ethereum smart contracts through a web interface.

SAMPLE VIVA VOCE QUESTIONS:

1. Describe the steps involved in developing a web interface for interacting with a deployed smart contract.
3. How does the web interface obtain the ABI and address of the deployed smart contract?
4. What are some common tasks that users can perform through the web interface to interact with the smart contract?
5. How does the web interface display contract data to the user, and how is it updated after interactions with the smart contract?

GAP ANALYSIS BY THE COURSE FACULTY /

COORDINATOR

In order to ensure the effectiveness and completeness of the blockchain lab modules, a gap analysis has been conducted to identify potential discrepancies between the current content and the desired learning outcomes. The following analysis highlights key areas where improvements or enhancements could be made to better align the modules with the intended objectives:

Understanding Blockchain Foundations:

Gap: Limited exploration of practical use cases for distributed computing and cryptography within blockchain applications.

Recommendation: Integrate real-world examples and case studies to illustrate how distributed computing and cryptography contribute to blockchain security and consensus.

Getting Familiar with the Ethereum Platform:

Gap: Inadequate emphasis on practical hands-on exercises for interacting with the Ethereum platform and its virtual machine.

Recommendation: Incorporate more interactive exercises and guided demonstrations to ensure students gain practical experience in using the Ethereum platform and its tools.

Introduction to Solidity Program Structure:

Gap: Insufficient coverage of the intricacies of Solidity program structure, compilation, and deployment environment.

Recommendation: Enhance content by providing detailed explanations of Solidity syntax, compilation steps, and deployment considerations to foster a deeper understanding.

Creating and Deploying a Simple Smart Contract:

Gap: Limited guidance on troubleshooting common issues that students might encounter during the process.

Recommendation: Include troubleshooting tips and solutions to common deployment challenges, empowering students to overcome obstacles independently.

Develop a Smart Contract with Setter and Getter Functions:

Gap: The absence of practical examples showcasing the potential use cases and benefits of setter and getter functions.

Recommendation: Provide real-world scenarios where setter and getter functions are crucial, highlighting their significance in data manipulation and interaction.

Develop Solidity Contracts for Inheritance and Polymorphism:

Gap: Insufficient emphasis on practical exercises that allow students to implement inheritance and polymorphism concepts.

Recommendation: Incorporate coding exercises that guide students through the process of implementing inheritance and polymorphism in Solidity contracts.

Familiarize with Remix Ethereum Tool:

Gap: Limited exposure to Remix Ethereum tool's advanced features beyond basic familiarity.

Recommendation: Introduce students to more advanced features of the Remix tool, such as debugging and contract analysis, to enhance their proficiency.

Setting Up a Simple Ethereum Wallet:

Gap: Inadequate coverage of security best practices for wallet setup and management.

Recommendation: Integrate a section on wallet security, emphasizing the importance of private key protection, two-factor authentication, and secure backup methods.

Hyperledger Fabric Demo:

Gap: Limited exploration of real-world use cases and scenarios where Hyperledger Fabric excels.

Recommendation: Introduce students to case studies illustrating how organizations have leveraged Hyperledger Fabric for enterprise blockchain solutions.

By addressing the identified gaps and incorporating the recommended enhancements, the blockchain lab modules can provide students with a more comprehensive and practical understanding of blockchain technology, its components, and its real-world applications. This approach ensures that students are well-prepared to apply their knowledge effectively in blockchain-related projects and initiatives.

Faculty name & Signature with date

HoD/BoS Chairman remarks:

.....
.....
.....

HoD/BoS chairman Signature with date