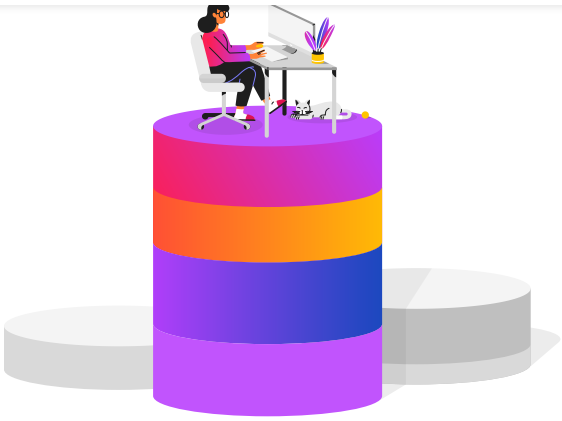

100 Solidity interview questions and answers in 2024

If you want to work as a successful Solidity developer in a top Silicon Valley company or build a team of brilliant Solidity developers, you've come to the right place. We have carefully prepared a list of Solidity developer interview questions for your Solidity interview to give you an idea of the type of Solidity interview questions you can ask or be asked.

Apply as Solidity developer

Last updated on Mar 31, 2024

Share this

Solidity, an object-oriented programming language, is in great demand. Based on languages such as Javascript, C++, and Python, developers can easily get started with it. There is tough competition among Solidity developers due to its increasing demand. Hence to crack the technical Solidity interview questions, we have given here a comprehensive list of questions segregated into basic, intermediate, and advanced.

BASIC SOLIDITY INTERVIEW QUESTIONS AND ANSWERS

1. Explain Solidity. What are the main components of a Solidity contract?

Hide Answer

Solidity is a statically-typed, high-level programming language primarily used to write smart contracts on Ethereum-based blockchain platforms. It was specifically designed to enable developers to easily create, deploy, and manage smart contracts on the Ethereum network. Solidity code is compiled to bytecode and executed on the Ethereum Virtual Machine (EVM), which ensures isolation, security, and transparency during contract execution.

The main components of a Solidity contract are:

Pragma statement: Specifies the required version of the Solidity compiler for the contract.

State variables: Variables that store the contract's state data persistently on the blockchain.

Constructor: A special function called once at contract deployment, used for initializing state variables.

Functions: Define the logic and behavior of the contract; can be divided into subcategories like view, pure, external, and internal functions.

Modifiers: Reusable code snippets that can be added to functions to modify their behavior, often to enforce access restriction or requirements.

Events: Custom data structures that emit transaction logs for external listeners to monitor contract activity and state changes.

Inheritance: Allows a contract to inherit properties (state variables, functions, events, and modifiers) from a base contract, enabling code reuse and

decentralized manner.

---

2. Define Ethereum smart contract.

Hide Answer

An Ethereum smart contracts is a self-executing, autonomous piece of code that runs on the Ethereum blockchain. It contains the terms of an agreement between parties, and it is enforced automatically by the Ethereum Virtual Machine (EVM) when certain predetermined conditions are met.

Smart contracts can store and manage data, transfer digital assets, execute functions, and interact with other contracts on the Ethereum network, enabling decentralized applications (DApps) to be built and executed without intermediaries. Smart contracts are typically written in a high-level programming language like Solidity and then compiled into bytecode for execution on the EVM.

---

3. Why is the Ethereum smart contract so special compared to other programs?

Hide Answer

Ethereum smart contracts are special compared to other programs due to the following reasons:

Decentralized execution: Smart contracts run on the Ethereum blockchain, which is a decentralized network. This means that the execution of a smart contract doesn't rely on a single authority or server, but rather on multiple nodes spread around the world. This increases the reliability and fault tolerance of the smart contract.

Immutable and tamper-proof: Once a smart contract is deployed on the Ethereum blockchain, its code cannot be altered. This makes smart contracts invulnerable to hacking or unauthorized changes, ensuring that the rules established in the contract remain unchanged.

Transparent and verifiable: All transactions and state changes in a smart contract are recorded on the Ethereum blockchain, and this information is

automatically when certain conditions specified in the contract are met. This eliminates the need for intermediaries and manual intervention to enforce the terms of an agreement, reducing the potential for human error or fraud.

Tokenization: Ethereum smart contracts can be used to create and manage digital assets through the use of tokens. This has led to various use cases in finance, gaming, collectibles, and more, that leverage tokenization as a way to represent unique assets or value.

---

4. Define Ethereum networks.

Hide Answer

Ethereum networks are blockchain ecosystems that are built using the Ethereum protocol. They are decentralized platforms that facilitate the execution of smart contracts and transactions using the Ether (ETH) cryptocurrency. There are primarily two types of Ethereum networks:

Mainnet: The mainnet is the primary and original Ethereum network. It is a public, decentralized, and permissionless network where Ether holds real-world value, and transactions and contract executions are performed using real Ether. The mainnet is used for deploying production-level smart contracts and applications, and all nodes maintain the network's security, validate transactions, and reach consensus following the Ethereum protocol.

Testnets: Testnets are alternative Ethereum networks used for testing and development purposes. They provide a sandbox environment where developers can deploy and interact with smart contracts without risking real Ether. Testnet Ether has no real-world value, and testnets are isolated from the mainnet to avoid any impact on the mainnet's security or stability. Some of the commonly used Ethereum testnets are Ropsten, Rinkeby, and Goerli.

---

5. Explain enum.

Hide Answer

Enums improve code readability by using descriptive names for states instead of using direct integer values, thus making the code easier to understand and maintain. In Solidity, enums are declared using the enum keyword followed by a descriptive name for the enumeration and a set of values enclosed in curly braces {}.

---

6. Explain the role of the Ethereum Virtual Machine (EVM).

Hide Answer

The Ethereum Virtual Machine (EVM) plays a crucial role in the Ethereum ecosystem, serving as the runtime environment for smart contracts running on the Ethereum blockchain. It ensures that smart contracts are executed securely and consistently across all nodes in the network. The main roles of the EVM can be summarized as follows:

Executing smart contracts: The EVM processes the EVM bytecode generated from high-level smart contract languages like Solidity. Each opcode in the bytecode represents basic operations like arithmetic, logic, and storage, which the EVM executes sequentially.

Isolating contract execution: EVM provides a sandboxed environment to execute smart contracts, ensuring that the execution is isolated from the underlying system and other contracts. This guarantees the safety and security of the network and prevents malicious contracts or bugs from affecting other parts of the system.

Maintaining state: The EVM is responsible for maintaining the state of the Ethereum blockchain, which includes the contract state and balances of all accounts. When a smart contract's state changes due to the execution of a function, the EVM updates the contract's state accordingly to ensure an accurate and up-to-date representation of the contract's data.

Handling gas: The EVM uses the concept of gas to measure the cost of executing a transaction or smart contract. Gas is paid in Ether and represents the computational resources required for executing a particular operation. The EVM

About us

Hide Answer

Gas is the price of gas in Ether that a user is willing to pay to execute a transaction on the network. Transactions with higher gas prices are prioritized by

I'm hiring developers I'm looking for jobs

Total gas fee = gasLimit * gasPrice per unit

Here gasLimit = maximum amount of gas that is going to spend on a single transaction

gasPrice = minimum gas cost of the execution of a transaction.

---

8. What is a gas limit in Solidity?

Hide Answer

A gas limit in Solidity refers to the maximum amount of gas a user is willing to spend on a transaction or a contract execution. When creating a transaction on the Ethereum network, users must specify a gas limit to ensure they don't accidentally spend more gas than they intend. If a transaction requires more gas than the specified gas limit, the transaction will be reverted, and used gas will not be refunded. Conversely, if a transaction requires less gas than the gas limit, the remaining gas will be returned to the user.

---

9. What is a variable in Solidity?

Hide Answer

State variables: These are the variables declared outside of any functions in a contract, and they are permanently stored on the Ethereum blockchain. Their values persist across function calls throughout the contract's lifecycle, and they can have different visibility levels: public, private, internal, and external.

Local variables: These are the variables declared within a function or a block, and their scope is limited to that function or block. They do not persist beyond the function call and are not stored on the blockchain.

Variables in Solidity also have data types, such as uint, int, address, bool, bytes, or string. Additionally, Solidity supports complex data structures like arrays, structs, and mappings.

---

10. What is the method of payment for gas?

Hide Answer

Gas is paid in ether using the formula: ether cost = gasPrice * gas. In this formula, the gas represents the gas cost of executing a transaction. gasPrice is in wei / gas, usually expressed in Gwei. A transaction also shows a gasLimit parameter- it specifies the maximum number of gas that a transaction can pay. A transaction without this could potentially deplete an account's Ether.

---

11. Give one difference between a uint8 and a uint16?

Hide Answer

Uint8 = store a number of up to 2^8 -1 (it has 8 bits)

Uint16 = store numbers of up to 2^16 – 1.

---

12. Give one difference between the state variable and the local variable?

Hide Answer

13. Who can read private and public variables?

Hide Answer

In Solidity, the visibility of variables determines who can access those variables.

Public variables: They can be accessed by any contract, function, or external entity. When a public state variable is declared in Solidity, the compiler automatically generates a getter function for that variable, which allows any external entity or other contracts to read its value. However, modifying the value of a public variable is still limited to the contract or derived contracts themselves.

Private variables: They can be accessed only within the contract that declares them. Private variables are not accessible by any external entity, nor by contracts derived from the contract in which they are declared. They provide encapsulation and help maintain the contract's internal state securely.

It's important to note that although private variables cannot be directly accessed by external entities, their values may still be read indirectly through transaction data on the blockchain, since all data is public on the Ethereum network. To ensure confidentiality, consider alternative approaches such as encrypting sensitive information off-chain.

---

14. Does the EVM understand Solidity?

Hide Answer

No. The Ethereum Virtual Machine (EVM) does not directly understand Solidity. Instead, it understands and executes bytecode, a lower-level language.

When you write a smart contract in Solidity, the code must go through a couple of steps before it can be executed on the EVM:

Compilation: The Solidity code is compiled into an intermediate representation called Ethereum bytecode using a compiler like solc. This bytecode is a sequence of low-level instructions that the EVM can understand.

and processes the corresponding bytecode instructions. Since EVM bytecode is a lower-level language, it is closer to machine code and more efficient for the EVM to execute, allowing for better performance and resource utilization.

---

15. What is a staking pool in Solidity?

Hide Answer

A staking pool in the context of Solidity is a smart contract that allows users to pool their cryptocurrency holdings (e.g., Ether or tokens) together and participate in various blockchain activities, like Proof-of-Stake (PoS) consensus mechanisms, liquidity provision on decentralized exchanges, or revenue-generating platforms like yield farming.

A staking pool is typically designed to provide users with proportional rewards based on their stake, maximize potential earnings, and mitigate risks associated with individual staking. The staking pool smart contract would define how users can add and withdraw their funds, how the rewards are calculated, and how they are distributed among the participants.

---

16. What is a proxy contract in Solidity?

Hide Answer

A proxy contract in Solidity is a design pattern that employs a secondary contract to act as an intermediary between the users and the main contract, which houses the core business logic. Proxy contracts are often used for upgradeability and storage optimization purposes.

The proxy contract maintains a reference to the main contract (often called the logic or implementation contract) and delegates calls from users to the main contract, effectively forwarding function calls and data to the appropriate methods in it. This allows developers to maintain the proxy contract's state, while the main contract's code can be replaced or updated without affecting the data.

A function is a group of instructions that perform a specific task. And it can be reused anywhere in the program, which saves the unreasonable use of memory and decreases the runtime of the program; by creating a function, users do not need to write the same code repeatedly.

---

18. Describe an event in Solidity.

Hide Answer

An event in Solidity is a custom data structure used to log information and notify external consumers, such as off-chain applications or services, about specific occurrences within a contract. Events serve as a convenient way to emit data that can be easily observed and monitored by external entities without reading the entire contract state.

Events are declared in contracts using the event keyword and usually contain parameters that define the data to be logged. When an event is triggered, the EVM emits a log that includes the event data, making it searchable and accessible via blockchain explorers or APIs.

---

19. What is the difference between public and private visibility in Solidity?

Hide Answer

In Solidity, visibility modifiers like public and private determine the accessibility of state variables, functions, and contracts. The difference between public and private visibility in Solidity is as follows:

Public: When a state variable, function, or contract is marked public, it is accessible from any contract (including the contract itself, derived contracts, and other external contracts) and can also be called directly by external transactions. For state variables, Solidity automatically generates a getter function to allow external access.

20. In Solidity, explain the constructor?

Hide Answer

In Solidity, a constructor is a special function within a contract that is called only once, at the time of contract deployment. The constructor is used to initialize the contract's state variables and set the initial state of the contract. It can be identified by the keyword constructor.

A Solidity constructor may have one or more arguments, allowing users to pass values during the contract deployment. Additionally, it can have visibility modifiers like public or internal. However, private and external modifiers are not allowed for constructors.

---

21. Explain delegatecall here in Solidity.

Hide Answer

delegatecall is a low-level function in Solidity used to call a function in another contract, while preserving the context (storage and caller address) of the calling contract. It allows the called function to access and modify the calling contract's storage, effectively enabling code reuse or upgradable contracts.

When delegatecall is used, the called function runs in the context of the calling contract, making it possible to modify state variables and access other functions of the calling contract as if they are in the same contract. This functionality should be used with caution due to potential security concerns and risks associated with giving another contract direct control over the calling contract's storage.

---

22. What is a library?

Hide Answer

contracts.

Functions in a library can be viewed as pure functions without knowledge of the contract's state. When a library function is called, the context of the calling contract remains unchanged, allowing library functions to operate on the calling contract's state variables through the use of the using keyword. This allows for efficient code reuse and helps contract developers modularize their smart contract implementation.

---

23. How many types of libraries are there in Solidity?

Hide Answer

Deployed- They have their own address, and several other smart contracts can use them.

Embedded- They don't have their own unique address. They are deployed as part of the code of the smart contract that uses them.

---

24. Mention two famous smart contract frameworks for Solidity.

Hide Answer

Truffle: Truffle is an Ethereum development framework that provides tools for building and testing smart contracts in Solidity. It simplifies the entire development process by offering a built-in testing environment, deployment management with migrations, and a powerful command-line interface. Truffle also supports popular Ethereum libraries such as Web3.js and provides access to Ganache, a personal blockchain for development and testing.

Hardhat: Hardhat is another leading smart contract development framework that focuses on developer experience and productivity. It offers a range of features, including an Ethereum development environment, a testing framework, and a task runner. Hardhat is well-known for Hardhat Network, an Ethereum node designed for local development, which supports advanced debugging and console.log() functionality directly from your Solidity contract. Additionally,

25. Mention two networks where you can deploy a Solidity smart contract.

Hide Answer

Two networks where you can deploy a Solidity smart contract are:

Ethereum Mainnet: The Ethereum Mainnet is the primary, public blockchain network where Ether has real-world value and smart contracts can interact with real users and decentralized applications.

Ropsten Testnet: Ropsten is a widely-used Ethereum test network that allows developers to deploy and test their smart contracts without incurring any real-world costs. It uses Proof of Work consensus mechanism and offers test Ether to simulate real-world conditions.

---

26. Can you list some distinctions between view and pure functions?

Hide Answer

Here are some distinctions between view and pure functions in Solidity:

State reading: view functions allow reading data from the contract's state, while pure functions do not access the contract's state at all.

State modification: Neither view nor pure functions modify the contract's state.

Gas consumption: Since view functions can access the contract's state, their gas consumption is dependent on the operations and data read from the state. On the other hand, pure functions do not access the state, so their gas consumption is more predictable as it depends only on the function's execution, not the state interactions.

Use cases: View functions are often used for retrieving information about the contract and its state, like checking a user's balance or getting some specific details from the contract. pure functions, in contrast, are used for computations that depend only on the function's input, like calculating a hash or performing a mathematical operation.

blockchain.

27. Please outline some differences between the Ethereum blockchain and Bitcoin.

Hide Answer

Ethereum Blockchain	Bitcoin
It is a blockchain platform, not a cryptocurrency that facilitates the trading of Ether.	Bitcoin is the first cryptocurrency that creates and enables the exchange of Bitcoins between multiple users.
Execute decentralized applications on the platform.	Created with the objective of only allowing the trading of bitcoins.
Consensus reached by POS algorithm. Faster computational time.	Consensus reached by POW algorithm. Slower computational time.

28. Outline some differences between a struct and an array?

Hide Answer

Collection of heterogeneous data.	Collection of homogeneous data.
Elements are referred to by their unique name.	Elements are referred to by subscripts.
It is a user-defined data type.	It is a derived data type.

---

29. What do you need to do to deploy a smart contract to the Ethereum network?

Hide Answer

To deploy a smart contract to the Ethereum network, follow these steps:

Write the smart contract: Create a Solidity smart contract, following best practices and conforming with the required standards, such as ERC-20 for tokens.

Test the smart contract: Thoroughly test the smart contract using tools like Remix IDE, Truffle, or Hardhat. Write unit tests and ensure your contract behaves as expected.

Compile the contract: Compile the smart contract using a Solidity compiler, such as solc, to generate bytecode and Application Binary Interface (ABI).

Choose a network: Determine the appropriate Ethereum network for deployment, such as the Ropsten or Rinkeby Testnets (for testing) or Mainnet (for production).

Fund your account: Ensure you have Ether in the account used for deployment. If using a testnet, request Ether from a faucet.

Prepare the deployment script: Create a deployment script or use a tool like Truffle or Hardhat to simplify the deployment process. Include the contract bytecode, ABI, constructor arguments (if any), and gas price in the script.

Unlock your wallet: Unlock your Ethereum wallet (private key), which is used for signing the deployment transaction.

Deploy the contract: Execute the deployment script or command. This will send a deployment transaction to the network, signed by your wallet.

Wait for transaction confirmation: Monitor the transaction hash for confirmation. Once the transaction is mined and confirmed, the smart contract address will be available.

About us

Looking for remote developer job at US companies?

Work at Fortune 500 companies and fast-scaling startups from the comfort of your home

Apply Now

INTERMEDIATE SOLIDITY INTERVIEW QUESTIONS AND ANSWERS

1. Show the Solidity smart contract layout.

Hide Answer

```
//statement for pragma (required)
pragma Solidity ^0.5.9;
//contract is declared (required)
contract B {
  //state variables
  uint b;
 //functions
  function foo() {...}
}
```

2. Are private variables really private?

Hide Answer

In Solidity, private variables are "private" within the context of the contract. This means that a private variable cannot be accessed or modified directly by other contracts or external entities.

private variables provide protection against unauthorized access within the contract code itself, they do not guarantee complete privacy from a determined observer.

---

3. **Explain when you would use an array versus a mapping.**

Hide Answer

When deciding whether to use an array or a mapping in Solidity, consider the following factors:

Use an Array when:

- You need to maintain a specific order of elements.
- You want to store a collection of elements that can be iterated over.
- You require a fixed or dynamic length data structure and need to know the total number of elements.
- You need to access elements using a numerical index.

Use a Mapping when:

- You need a key-value structure for associating one data type with another.
- You want fast and constant-time lookups of elements irrespective of the mapping size.
- You don't need to know the total number of elements or maintain their order.
- You want to avoid duplicate keys, as each key in a mapping can have only one associated value.
- You want a data structure that doesn't necessarily have a predefined or fixed size.

In summary, choose an array when you need a collection of data that preserves the order and allows iteration. Opt for a mapping when you need an efficient key-value store that avoids duplicates and maintains a constant-time lookup.

In Solidity, memory and storage are two different types of data locations. The key difference between them is as follows:

memory: A temporary storage location that exists only for the duration of a function call. Data stored in memory is not persisted between function calls and is much cheaper, in terms of gas costs, than storage.

storage: A permanent storage location that exists on the Ethereum blockchain, where contract state variables are stored. Data stored in storage persists across function calls and is more expensive, in terms of gas costs, compared to memory.

---

5. What are Solidity modifiers?

Hide Answer

Solidity modifiers are reusable pieces of code that can be added to functions to modify their behavior. They typically serve the purpose of validating conditions or adding access control restrictions to functions. Modifiers can be attached to a function using an annotation-like syntax, and their code is executed before the main function code.

Modifiers can make code more readable, maintainable, and concise by abstracting away repetitive logic. When a function with a modifier is called, the modifier's code is executed first, and upon satisfying the required conditions, the original function's code is executed.

---

6. What is a fallback function in Solidity?

Hide Answer

A fallback function in Solidity is a special, unnamed function that gets executed whenever a contract receives Ether without any data or when a function call is made without specifying any function. It serves as the default function for a

Hide Answer

A decentralized exchange (DEX) is a cryptocurrency exchange that operates without relying on a centralized authority or intermediary, such as a traditional exchange or a bank, to facilitate and manage trades. Instead, DEXs run on a network of nodes, usually powered by blockchain or distributed ledger technology, allowing traders to retain control over their funds and private keys.

DEXs utilize smart contracts, often developed in Solidity, to automate trade execution, manage order books, and enable various trading pairs. Users trade directly from their wallets, removing the need to deposit funds into a centralized platform. This contrasts with centralized exchanges that require users to deposit funds into their platform's wallets, exposing users to potential hacks or loss of funds.

---

8. What is the difference between assert and require in Solidity?

Hide Answer

Assert function	Require function
Assert function is generally used when users have to check or test the invariants in the existing code.	This function is used to ensure valid conditions, such as inputs, or to validate return values or contract state variables are met from calls to external contracts.

---

9. Give the name of 3 data types that you use often?

Hide Answer

and loops.

address: This data type is used to store Ethereum addresses. It is often used when working with account addresses and contract instances, as well as for performing Ether transfers and contract calls.

string: A dynamic array of characters, mainly used for storing and manipulating text data or any arbitrary-length sequence of bytes. It is useful for storing information like user names, descriptions, or any other textual data.

---

10. In Solidity, how can you declare an array of integers?

Hide Answer

In Solidity, you can declare an array of integers by specifying the data type followed by square brackets []. For example, to declare an array of unsigned integers, you would use the uint[] data type. You can initialize the array while declaring it or create an empty array.

Here are two examples of how to declare an array of integers in Solidity:

Declare and initialize an array of unsigned integers:

```
pragma solidity ^0.8.0;

contract IntegerArray {
    // Declaring and initializing an array of unsigned integers
    uint[] public numbers = [1, 2, 3, 4, 5];
}
```

Declare an empty array of unsigned integers

:

---

11. How can you map addresses to booleans in Solidity?

Hide Answer

```
pragma solidity ^0.8.0;

contract AddressBooleanMapping {
    // Declare a mapping that maps addresses to booleans
    mapping(address => bool) public addressToBool;

    // Set a boolean value for a given address
    function setBoolForAddress(address _address, bool _boolValue) public {
        addressToBool[_address] = _boolValue;
    }

    // Get the boolean value of a given address
    function getBoolForAddress(address _address) public view returns (bool) {
        return addressToBool[_address];
    }
}
```

---

12. Write the code to add data to an array that has been declared as a state variable?

Hide Answer

Here's an example of a Solidity contract where we declare an array of unsigned integers as a state variable and define a function to add data to the array:

```
pragma solidity ^0.8.0;

contract ArrayExample {
    // Declare an array of unsigned integers as a state variable
    uint[] public myArray;

    // Function to add data to the array
    function addToMyArray(uint value) public {
        // Add value to the end of the array
        myArray.push(value);
    }
}
```

the state of the contract, thus persisting the array's data changes on the blockchain.

---

13. How can you add data to a mapping which is declared as a state variable?

Hide Answer

In Solidity, a mapping declared as a state variable is a key-value storage type that allows you to map keys to corresponding values. Here's an example of how you can declare a mapping as a state variable and then add data to it within a contract:

```
pragma solidity ^0.8.0;

contract SimpleMapping {
    // Declare a mapping as a state variable
    mapping(uint => string) public identification;

    // Function to add data to the mapping
    function setIdentifier(uint id, string memory name) public {
        identification[id] = name;
    }
}
```

In this example, the identification mapping is declared as a state variable, mapping uint keys to string values. The setIdentifier function takes an id (uint) and a name (string) as input, and assigns the name to the id in the identification mapping. This adds the key-value pair to the mapping.

---

14. Tell me about smart contract's ABI?

Hide Answer

A smart contract's ABI, or Application Binary Interface, is a JSON representation of the contract's methods and events. It serves as an interface between the

- Function names
- Function input and output types
- Function modifiers (like payable, view, or pure)
- Event names
- Event input types and indices

With the help of the ABI, a client-side application can construct correct data types, encoding, and decoding of function calls and events when interacting with a smart contract on the Ethereum blockchain.

---

15. Give some reasons why you need a private variable in your code?

Hide Answer

There are several reasons why you might want to use private variables in your Solidity code:

Encapsulation: Private variables help maintain a clear separation between internal state and external interactions. By keeping the internal workings of a contract hidden, you can prevent external manipulation or unintended access to sensitive information.

Security: Private variables can prevent unauthorized access to contract data, which is particularly important when handling sensitive or valuable information like user balances, ownership details, or internal contract state.

Upgradability: When contract state variables are kept private, you can more easily introduce logic updates or bug fixes without affecting external interfaces, maintaining backward compatibility.

Code maintainability: Private variables limit the visibility and accessibility of your contract's state, which helps to identify unintended external dependencies and ensures that any potential modifications to the contract's state are done through the provided contract functions.

16. Write the 2 APIs used to interact with a smart contract?

Hide Answer

eth_sendTransaction (transaction): eth_sendTransaction is a JSON-RPC API method provided by Ethereum nodes to create and broadcast transactions. JSON-RPC (Remote Procedure Call) is an API protocol that enables client-server communication over HTTP or WebSocket connections. Ethereum implements JSON-RPC to allow developers to interact with the Ethereum network and its nodes, including sending transactions and querying blockchain data.

eth_call (call): eth_call is an API method provided by Ethereum nodes. It is part of the JSON-RPC API suite, which allows developers to interact with the Ethereum blockchain over HTTP or WebSocket connections.

eth_call is an API method provided by Ethereum nodes. It is part of the JSON-RPC API suite, which allows developers to interact with the Ethereum blockchain over HTTP or WebSocket connections.

---

17. List 4 famous Ethereum wallets

Hide Answer

Here are four popular Ethereum wallets:

Metamask: A browser extension and mobile wallet that supports Ethereum and Ethereum-based tokens (such as ERC-20 and ERC-721). Metamask enables users to interact with decentralized applications (dApps) straight from their web browser or smartphone.

MyEtherWallet (MEW): An open-source, client-side wallet that allows users to create new Ethereum wallets, manage existing ones, and interact with smart contracts. MEW is accessible through a variety of platforms, including web browsers and mobile apps.

Ledger: A hardware wallet that provides offline storage and enhanced security for cryptocurrencies, including Ethereum and Ethereum-based tokens. Ledger wallets, such as the Ledger Nano S and Ledger Nano X, support interaction with

cryptocurrencies offline. Trezor wallets, such as Trezor One and Trezor Model T, can also connect with Ethereum dApps and smart contracts through integrated wallet software like Metamask and MyEtherWallet.

---

18. List some ways you can instantiate a struct?

Hide Answer

In Solidity, you can instantiate a struct in several ways. Here are a few of them:

- Declare a new instance using the StructName and initialize it by assigning values to all its properties inside parentheses:
  StructName memory instanceName = StructName({property1: value1, property2: value2});
- Declare a new instance using the StructName and initialize it by assigning values in the same order as defined in the struct declaration:
  StructName memory instanceName = StructName(value1, value2);
- When creating a nested struct, you can use the same approaches mentioned above:
  ParentStructName memory instanceName = ParentStructName({
  property1: value1,
  nestedStruct: StructName({property1: value1, property2: value2})
  });

Remember that if you want to create a struct within a function, you need to use the memory keyword when declaring it. On the other hand, if you need to create a struct instance as a state variable, you don't need to use memory.

---

19. How can you instantiate a struct with inner mapping?

Hide Answer

In Solidity, you cannot directly instantiate a struct with an inner mapping because mappings cannot be copied or initialized. However, you can work around this

```
pragma solidity ^0.8.0;

contract StructWithMapping {

    struct Item {
        uint256 id;
        string name;
        mapping(address => uint256) userRatings;
    }

    Item[] public items;

    function createItem(string memory _name) public {
        Item memory newItem;
        newItem.id = items.length;
        newItem.name = _name;
        items.push(newItem);
    }

    function rateItem(uint256 _itemId, uint256 _rating) public {
        require(_rating >= 1 && _rating <= 5, "Rating should be between 1 and 5.");
        require(_itemId < items.length, "Item not found.");
        items[_itemId].userRatings[msg.sender] = _rating;
    }

}
```

---

20. How can you join array and mapping to allow iteration and struct lookup?

Hide Answer

In Solidity, arrays and mappings cannot be directly connected. However, you can achieve iteration and struct lookup by using a combination of an array, mapping, and a struct. Here's an example to demonstrate this:

```
    uint id;
    string content;
  }

  Data[] public dataList;
  mapping (uint => uint) public dataIndices;

  function addData(uint _id, string memory _content) public {
    // Assign the index of the data in dataList array to the _id
    dataIndices[_id] = dataList.length;

    // Add the data to dataList array
    dataList.push(Data(_id, _content));
  }

  function removeData(uint _id) public {
    // Retrieve the index of the data in dataList array
    uint index = dataIndices[_id];

      // Remove the data from dataList array by swapping it with the last element and then
removing the last element
    dataList[index] = dataList[dataList.length - 1];
    dataList.pop();

    // Update the dataIndices mapping for the swapped Data element
    dataIndices[dataList[index].id] = index;
  }

  function getData(uint _id) public view returns (Data memory) {
    // Get the Data from dataList array using the index from dataIndices mapping
    return dataList[dataIndices[_id]];
  }
}
```

In this example, we store the data in a struct called Data. To allow iteration and struct lookup, we use an array dataList to store the Data structs, and a mapping dataIndices to link the id to the index of the data in the dataList array. By doing this, you can iterate through the dataList array, and also use the dataIndices mapping to quickly find the index of a specific Data struct in the dataList array.

21. For an in-memory array, how to add a value?

Hide Answer

In Solidity, to add a value to an in-memory array, you need to make sure the array size is large enough to hold the new value. Unfortunately, memory arrays do not

Here's an example that demonstrates how to add a value to an in-memory array by copying the data into a new array:

```
pragma solidity ^0.8.0;

contract MemoryArray {

    function addValue(uint[] memory oldArray, uint newValue) public pure returns (uint[] memory newArray) {
        newArray = new uint[](oldArray.length + 1); // Create new array with one more slot

        for(uint i = 0; i < oldArray.length; i++) {
            newArray[i] = oldArray[i]; // Copy old array elements to the new array
        }

        newArray[oldArray.length] = newValue; // Add new value to the last slot

        return newArray;
    }
}
```

---

22. What is the difference between an ERC-20 and ERC-721 token?

Hide Answer

ERC-20	ERC-721
fungible tokens	non-fungible tokens
It represents a specific asset or utility	It represents a unique asset or asset class.

---

23. Explain reentrancy attack?

previous call is finished. It often takes advantage of the contract's functions that make external calls to untrusted contracts while still having control over the state variables.

In a reentrancy attack, the malicious contract hijacks the control flow and can potentially drain the funds or exploit other vulnerabilities in the targeted contract.

---

24. What is a hard fork in Solidity?

Hide Answer

A hard fork is a change to the underlying consensus rules or protocol that results in the creation of a new, independent blockchain branch. The term "hard fork" is used to refer to the permanent divergence in the blockchain network due to incompatibilities between the old rules and the new rules.

In the context of Ethereum, a hard fork may occur when there is a change to the Ethereum protocol that requires all participants in the network (miners, nodes, and clients) to update their software. Participants who do not upgrade will remain on the old chain, while those who do will create a new chain with the updated rules.

---

25. What is the difference between a requirement and a revert statement in Solidity?

Hide Answer

In Solidity, both require() and revert() statements are used to handle errors and exceptions by checking for conditions and reverting the transaction if the conditions are not met. However, there are differences in their use cases and behavior, as detailed below:

require() function: require() is mainly used to validate input parameters and preconditions before executing functions or contracts. It checks if a specified condition is true and, if not, reverts the transaction along with a custom or default

back any state changes made during the execution. revert() is typically used to indicate that an error or exception has occurred while executing the contract, or a specific condition that should not have been reached has been encountered.

---

26. What is a token standard in Solidity?

Hide Answer

A token standard in Solidity is a set of rules and guidelines, including a predefined interface, that governs how tokens should be implemented on the Ethereum blockchain. By following a token standard, developers create tokens that are compatible with different wallets, contracts, and other services within the Ethereum ecosystem. These standards define the methods and events that a token contract should include, enabling the interoperability of different tokens throughout the network.

---

27. What is a privacy token in Solidity?

Hide Answer

A privacy token in Solidity refers to a token/asset implemented in a smart contract on the Ethereum blockchain with enhanced privacy features. Privacy tokens aim to protect the identity of token holders and keep transaction details confidential when compared to standard tokens, such as ERC20 or ERC721.

The primary objective of privacy tokens is to enable private transactions, enhanced security, and reduced traceability while still ensuring compliance with decentralized networks and ecosystems.

---

28. What is a governance token in Solidity?

Hide Answer

A governance token in Solidity is a token that represents the voting rights and decision-making power within a decentralized network or platform. Governance

29. What is a wrapped token in Solidity?

Hide Answer

A wrapped token in Solidity is a tokenized version of a digital asset, typically implemented as a smart contract on the Ethereum blockchain. Wrapped tokens are created to represent the value of various cryptocurrencies or assets while enabling compatibility with the Ethereum network, enhancing functionality and interoperability within the Ethereum ecosystem.

Probably the most well-known example of a wrapped token is the Wrapped Bitcoin (WBTC) - an ERC20 token that represents Bitcoin on the Ethereum blockchain. Each WBTC token has the same value as one Bitcoin and is backed by actual Bitcoin held in a custodial wallet. This allows users to interact with Bitcoin within the Ethereum DeFi ecosystem while maintaining the value of the native asset.

30. What is a rollup?

Hide Answer

A rollup is a Layer-2 scaling solution built on top of existing blockchain networks, such as Ethereum, to improve the throughput and reduce the transaction costs. Rollups function by bundling or "rolling up" multiple transactions off-chain into a single transaction using a cryptographic proof, hence the name "rollup." This proof is then submitted and verified on the base blockchain, which reduces the number of on-chain transactions while maintaining the security guarantees provided by the underlying blockchain.

31. What is a soft fork in Solidity?

Hide Answer

including Ethereum, which Solidity is built upon.

For instance, a change in the Ethereum network's consensus algorithm, applied through a soft fork, would still allow nodes that have not upgraded to the new version to validate and accept blocks generated by nodes using the updated rules. However, the nodes that have not upgraded might still produce blocks that may be considered invalid by up-to-date nodes.

---

32. What is a decentralized application (dApp) in Solidity?

Hide Answer

A decentralized application (dApp) is a distributed application built on top of a blockchain network, like Ethereum. The term "in Solidity" refers to decentralized applications whose smart contracts are written in the Solidity programming language.

dApps leverage smart contracts, which are self-executing contracts with the terms directly written into code, to create a transparent and trustless environment. The main features of decentralized applications are:

Open source: dApps have their source code available to everybody, promoting transparency.

Decentralized: dApps are built on top of blockchain networks that store data on multiple nodes, reducing the risk of a single point of failure and central control.

Consensus-driven: dApps use cryptographic techniques and consensus mechanisms that incentivize participants to behave honestly and fairly.

No central authority: dApps operate autonomously without intermediaries, providing censorship resistance and reducing dependency on third parties.

---

33. Give one difference between an event and a function in Solidity.

Hide Answer

about state changes or important occurrences within the contract.

On the other hand, a function is a piece of code that can be called by external components or other functions within the contract to manipulate data or execute some logic. Functions are the core building blocks of a contract and can read from or write to the contract's state, transfer ether, interact with other contracts, and perform various other actions.

In summary, a key difference between an event and a function in Solidity is that an event is used for logging and broadcasting information whereas a function is used for executing logic and manipulating data within the contract.

---

34. How to get the list of all keys in a mapping (like object.keys() in Javascript)?

Hide Answer

Using object.keys(), you can get all the keys of an object. It takes an argument and returns an array for all the keys. It gives the object's own enumerable string-keyed property names.

---

35. List 3 mechanisms for code reuse here.

Hide Answer

In Solidity, three mechanisms for code reuse are:

Functions: Functions are reusable units of code within a single contract. You can define functions that perform specific tasks or calculations and call these functions from other parts of the contract whenever needed. Functions help make the code manageable and modular.

Inheritance: Solidity allows contracts to inherit properties (state variables, functions, events, and modifiers) from other contracts using inheritance. Inheritance promotes the reuse of existing, well-defined code and prevents the need to define similar functionalities in multiple contracts. Inherited contracts are

```
contract BaseContract {
    function doSomething() public {
        // Some logic
    }
}

contract DerivedContract is BaseContract {
    function callDoSomething() public {
        doSomething(); // Call function from the BaseContract
    }
}
```

Libraries: Solidity libraries are reusable pieces of code that can be deployed separately and then linked to contracts. Libraries can define reusable functions and utilize their own internal state, but they cannot hold any state data or interact with Ether. Contracts can call the functions defined within a library, allowing for code reuse and optimization. Solidity also supports using the using keyword, which allows the library's functions to be called as if they were contract methods.

Example:

```
library SafeMath {
    function add(uint x, uint y) internal pure returns (uint) {
        uint sum = x + y;
        require(sum >= x, "SafeMath: addition overflow");
        return sum;
    }
}

contract MathContract {
    using SafeMath for uint;

    function add(uint x, uint y) public pure returns (uint) {
        return x.add(y); // Use the SafeMath library's add function
    }
}
```

---

36. How can you make one contract inherit from the other?

variables, functions, events, and modifiers from the base contract. Inheritance can be used to create more modular, reusable, and organized code.

Here's an example to demonstrate contract inheritance in Solidity:

```
pragma solidity ^0.8.0;

// Base contract
contract Base {
    uint public x;

    function setX(uint _x) public {
        x = _x;
    }

    function getX() public view returns (uint) {
        return x;
    }
}

// Derived contract
contract Derived is Base {
    uint public y;

    function setY(uint _y) public {
        y = _y;
    }

    function getY() public view returns (uint) {
        return y;
    }
}
```

In this example, Derived contract inherits from Base contract using the is keyword. As a result, the Derived contract can access and use the x state variable, setX() function, and getX() function from the Base contract. The Derived contract also has its own state variable y and functions setY() and getY() for additional functionality.

---

37. Is it compulsory to make an address “address payable” to transfer ERC20 tokens?

Hide Answer

tokens between addresses without requiring "address payable".

To transfer ERC20 tokens, you will need to interact with the respective ERC20 token contract using the provided transfer() or transferFrom() (combined with approve()) functions, which work with regular non-payable addresses.

---

38. What's new with Solidity 0.5.x vs. 0.4.x?

Hide Answer

- Add memory storage keywords to private, public, and internal complex parameters that don't have predefined storage locations.
- For fallback functions "add function" visibility of external
- Change function contractName to constructor
- Change constant function state mutability modifier to view

---

39. Give three ways to save gas.

Hide Answer

To make transactions more efficient and save gas in Solidity, developers can use different optimization techniques. Here are three common ways to save gas:

Use less expensive operations: Some operations in Solidity consume less gas than others, so it is important to select the right operations whenever possible. For example, use +=, -= instead of repeated + and - operations, and use ++ and -- increment/decrement operators to modify values. Additionally, using bitwise operations (&, |, ^) whenever possible instead of arithmetic operations can lead to gas savings.

Optimize storage: Setting a storage value costs more gas if it changes from zero to non-zero and less for a non-zero to non-zero change. To optimize, you can implement two approaches:

data types within struct efficiently to minimize the gas cost. Thus, organizing state variables within struct can save gas.

Reuse code with libraries: Extensive usage of functions, loops, and conditions can increase gas consumption. Using libraries for frequently used functions can help you to reduce the amount of bytecode and save gas. Also, consider using internal functions that are inlined when called, earlier in the contract to reduce the overall size of compiled bytecode.

---

40. Write in an order uint128, bytes32, and another uint128 to save gas.

Hide Answer

When organizing the order of data types in a Solidity contract to save gas, you should consider the concept of packing variables. This is especially relevant when storing multiple smaller data types. By placing smaller data types next to each other, you can utilize the available space within a single storage slot, which is 32 bytes wide, and save gas. The example below demonstrates how to organize uint128, bytes32, and another uint128 in an efficient order:

```
pragma solidity ^0.8.0;

contract GasOptimizer {
    uint128 variableA;
    bytes32 variableB;
    uint128 variableC;

    function setValues(uint128 _a, bytes32 _b, uint128 _c) public {
        variableA = _a;
        variableB = _b;
        variableC = _c;
    }
}
```

In the above example, variableA and variableC are both uint128, which effectively occupy 16 bytes each. Since a storage slot is 32 bytes wide, these two variables combined total 32 bytes, which fit into a single storage slot. variableB is a bytes32,

Hide Answer

The ABIEncoderV2 pragma statement is an experimental feature in Solidity that enables the more advanced ABI (Application Binary Interface) encoder and decoder functionalities, extending the capabilities of the Standard ABI Encoder (ABIEncoderV1). In particular, ABIEncoderV2 supports struct types and nested arrays as function parameters and return values.

To enable the use of ABIEncoderV2, you need to include the following pragma statement in your Solidity file:

pragma experimental ABIEncoderV2;

---

42. What is a decentralized identifier (DID) in Solidity?

Hide Answer

A Decentralized Identifier (DID) is not specific to Solidity; it is a general concept that can be implemented in any smart contract or blockchain platform. A DID is a new type of global identifier that enables self-sovereign identity by being generated and controlled by the identity owner, rather than relying on a centralized authority. DIDs are typically associated with verifiable credentials and decentralized identity systems like Decentralized Public Key Infrastructure (DPKI).

In the context of Solidity, a DID can be implemented by creating a smart contract that handles the registration, storage, and management of these identifiers on the Ethereum blockchain. For example, using Ethereum-based standards such as ERC725 (identity) and ERC735 (claims) to create a decentralized identity contract.

Looking for remote developer job at US companies?

Work at Fortune 500 companies and fast-scaling startups from the comfort of your home

ADVANCED SOLIDITY INTERVIEW QUESTIONS AND ANSWERS

1. Write two kinds of assembly?

Hide Answer

There are two kinds of assembly in Solidity: Inline Assembly and Stand-Alone Assembly.

Inline Assembly

Inline Assembly is a low-level, EVM-specific assembly language used within Solidity smart contracts to optimize the execution of specific code sections or access low-level EVM operations that are not directly available in Solidity. It is encapsulated within Solidity code using the assembly keyword.

Example of Inline Assembly in Solidity:

```
pragma solidity ^0.8.0;

contract InlineAssemblyDemo {
    function getAddition(uint256 a, uint256 b) public pure returns (uint256 result) {
        assembly {
            result := add(a, b)
        }
    }
}
```

In this example, the getAddition() function uses Inline Assembly to perform addition with the low-level add opcode of the EVM.

Stand-Alone Assembly (Yul):

Stand-Alone Assembly, also known as Yul or JULIA (intermediate language), is an intermediate language used by the Solidity compiler as an intermediate step between the high-level code and the actual bytecode produced for the EVM. Yul is designed to be simple, efficient, and easy to optimize. It is intended to serve as a common base for higher-level high-level languages and is usable as an

```
{
    let x := calldataload(0)
    let y := calldataload(32)
    let sum := add(x, y)
    mstore(0, sum)
    return(0, 32)
}
```

This Yul code snippet takes in two 256-bit values from the calldata, adds them, and returns the result. Note that this code cannot be used directly inside a Solidity contract, but illustrates the syntax and simplicity of a Stand-Alone Assembly.

---

2. How to protect against a reentrancy attack?

Hide Answer

To protect against a reentrancy attack, follow these best practices:

Use the Checks-Effects-Interactions pattern: Arrange your contract's code in the following order - perform input checks, update state variables, and lastly, interact with other contracts. This ensures that the state variables are up-to-date before any external calls.

```
function withdraw() public {
    uint256 amount = balances[msg.sender];

    // Checks
    require(amount > 0, "Nothing to withdraw");

    // Effects
    balances[msg.sender] = 0;

    // Interactions
    (bool success, ) = msg.sender.call{value: amount}("");
    require(success, "Transfer failed");
}
```

Use function modifiers: Create a function modifier that prevents the reentrancy attack by not allowing the function to be called again until the previous execution

```
modifier noReentrancy() {
    require(!locked, "Reentrant call");
    locked = true;
    _;
    locked = false;
}

function withdraw() public noReentrancy {
    // Your withdrawal logic here
}
```

Use the transfer() function: The transfer() function is a safer method for sending ether because it only forwards a limited amount of gas to the recipient. This is usually not enough gas for the recipient to execute arbitrary code, thus preventing reentrancy attacks. However, this method is now discouraged in favor of call{value: amount} due to new gas mechanics introduced by EIP-1884.

```
function withdraw() public {
    uint256 amount = balances[msg.sender];
    require(amount > 0, "Nothing to withdraw");

    balances[msg.sender] = 0;
    msg.sender.transfer(amount);
}
```

Use Pull Payment (i.e., Withdraw Pattern): Instead of pushing payments from the contract to the recipients, allow recipients to withdraw their funds from the contract. This method separates the accounting logic from the actual transfer of funds, reducing the attack surface for reentrancy attacks.

```
    uint256 amount = pendingWithdrawals[msg.sender];
    require(amount > 0, "Nothing to withdraw");

    // Update pendingWithdrawals before transferring to avoid reentrancy
    pendingWithdrawals[msg.sender] = 0;

    // Transfer the amount
    (bool success, ) = msg.sender.call{value: amount}("");
    require(success, "Transfer failed");
}
```

---

3. Is it feasible to send a transaction without having to charge customers for gas?

Hide Answer

You'll often find this question among the Solidity developer interview questions, and the answer is Yes. You'd have people sign a message on the front end first. The message and signature would then be routed to a centralized backend (off-chain) that would establish a transaction and include the payload (message + signature).

This means that instead of the user's wallet, the app's wallet will cover gas costs. A smart contract will validate the signature's validity and perform an activity on behalf of the user on the blockchain.

---

4. What is the use of the payable keyword in Solidity?

Hide Answer

The payable keyword in Solidity is used as a modifier for a function or a fallback method to indicate that the function can receive Ether directly. When a function is marked as payable, it enables the contract to accept Ether directly as part of the transaction calling the function.

If a function is not declared as payable, sending Ether to the function will result in a runtime error, causing the transaction to revert.

```
    uint256 public totalReceived;

    function sendEther() public payable {
        totalReceived = totalReceived + msg.value;
    }

    function getBalance() public view returns (uint256) {
        return address(this).balance;
    }
}
```

In this example, the sendEther function is marked as payable, allowing it to accept Ether payments and increment the totalReceived counter.

---

5. What is the equivalent of a Javascript console.log in Solidity for debugging?

Hide Answer

This question is often a part of the Solidity developer interview questions. In Solidity, there isn't a direct equivalent to console.log in JavaScript. Instead, you can use events for debugging purposes. You can define an event and emit it with the values you want to log. External clients or applications can then watch for this event and process the logged data.

Here's an example:

```
pragma solidity ^0.8.0;

contract DebuggingExample {
    event Log(string message, uint value);

    function exampleFunction(uint x) public {
        // Some code logic
        emit Log("The value of x is:", x);
    }
}
```

In this example, you would watch the Log event in your client application to observe the emitted message and value. Keep in mind that using events for

6. What do you understand by PoW consensus?

Hide Answer

PoW, or Proof of Work, is a consensus mechanism used in blockchain technology to validate and confirm new transactions and create new blocks. It requires participants, known as miners, to perform complex and resource-intensive calculations to solve a cryptographic puzzle. When a miner solves the puzzle, they can propose the next block to be added to the blockchain.

The PoW consensus mechanism helps to ensure that the blockchain remains secure and decentralized, as it makes it difficult for any single participant to monopolize the computational resources or perpetrate malicious actions. Networks such as Bitcoin and Ethereum use PoW as their primary consensus mechanism. However, concerns regarding energy consumption and environmental impact have led to the development of alternative consensus mechanisms such as Proof of Stake (PoS) and Delegated Proof of Stake (DPoS).

---

7. **Explain consensus algorithm's function?**

Hide Answer

A consensus algorithm is a fundamental component of a blockchain network that enables its nodes or participants to reach an agreement on the validity and authenticity of transactions and data. The main functions of a consensus algorithm include:

Transaction validation: Consensus algorithms ensure that only valid and legitimate transactions are added to the blockchain. By validating each transaction, the algorithm helps to maintain the integrity and security of the network.

Agreement on the state of the blockchain: Consensus algorithms facilitate agreement among nodes on the current state of the blockchain, which is necessary to maintain a consistent and up-to-date ledger.

Appending new blocks: The algorithms provide a mechanism for proposing and appending new blocks to the blockchain. These blocks contain recent

continue to function correctly. This contributes to a blockchain network's resilience and reliability.

Security: Consensus algorithms are designed to prevent malicious activities such as double-spending and other attacks on the network. They make it difficult for any single participant or a group of participants to take control of the network or compromise its integrity.

---

8. **An abstract contract is preferable over an interface in Solidity. Why?**

Hide Answer

Actually, it's not accurate to say that an abstract contract is always preferable over an interface in Solidity. Both abstract contracts and interfaces serve different purposes, and whether to use one over the other depends on the specific use case.

Interface:

- Interfaces are used to define the external functions that a contract is required to implement, enabling a consistent way to interact with other contracts.
- Interfaces can only contain function signatures without implementation (no function bodies).
- Interfaces cannot have state variables or constructors.
- Interfaces support multiple inheritance, letting a contract inherit from multiple interfaces.

Abstract Contract:

- Abstract contracts are used as a base for other contracts to inherit from, but they can't be deployed on their own because they contain at least one unimplemented function.
- Abstract contracts can have implemented functions, state variables, constructors, and events.
- Abstract contracts also support inheritance, yet unlike interfaces, they can have constructors and state variables.

interact with other contracts, then an interface is the better choice. However, if you need a base contract that can include common functionality, state variables, and constructor logic that other contracts can inherit from, then an abstract contract would be more suitable.

---

9. Mention some restrictions on enumeration use?

Hide Answer

Enumerations in Solidity are a convenient way to work with a fixed set of constant values. Despite their usefulness, they also come with some restrictions:

Integer conversion: Enumeration values are automatically assigned integer values starting from 0, but you cannot directly convert or assign numbers to enumeration types. An explicit cast is necessary to convert from integer values to the enumeration type.

Limited scope: Enumerations can only be declared at the contract level, and they cannot be defined inside a function or struct. This may limit their usability in certain situations.

Limited comparison operators: Enumerations support only basic comparison operators (== and !=). They do not support the greater-than (>), less-than (<), greater-than-or-equal-to (>=), or less-than-or-equal-to (<=) comparison operators. However, their underlying integer values can be compared using these operators after explicitly casting the enumeration value to an integer.

No dynamic enumeration: Enumerations in Solidity are static and require a predefined set of constants. They cannot be created or modified dynamically during runtime.

No arithmetic operations: Enumerations do not support arithmetic operations such as addition or subtraction. To perform arithmetic operations, you need to explicitly cast the enumeration value to an integer type and then perform the operations on the integer values.

---

10. What is the indexed keyword in the event definition?

event parameter is marked as indexed, it becomes a part of the indexed log's data structure known as "topics".

There can be up to three indexed parameters in an event, and these parameters will be stored as separate topics alongside the event signature in the log. Non-indexed parameters, on the other hand, are stored together in the log's data section.

With indexed parameters, clients like external applications or blockchain explorers can easily and efficiently search, filter, or listen to specific events based on specified indexed values. This is particularly useful when monitoring a large number of logs or looking for specific information within a blockchain.

---

11. What is a Merkle tree in Solidity?

Hide Answer

A Merkle tree, or binary hash tree, is a data structure used in computer science and cryptography for efficiently verifying the contents of large data sets. Although Merkle trees are not a part of Solidity as a language, they play a significant role in Ethereum, the blockchain platform where Solidity contracts are predominantly deployed.

In Ethereum, a Merkle tree is used to store the data of transactions (Trie), state (State Trie), and storage (Storage Trie) efficiently and securely. The tree consists of leaves and nodes, where the leaves contain the actual data (e.g., transactions or storage), and each non-leaf node stores the hash of its child nodes. The root of the tree, known as the Merkle root, represents the hash of all the tree's elements and is included in each block header to verify the block's transactions.

---

12. What is a sidechain in Solidity?

Hide Answer

A sidechain is not directly related to Solidity as a language; however, it is a concept used in blockchain technology. Sidechains are separate, individual

The purpose of using a sidechain is to address issues like network congestion, performance, and scalability on the main blockchain. The sidechain can process transactions and execute smart contracts independently from the main chain, thus distributing the workload and reducing the transaction processing burden on the primary blockchain. This can lead to faster transaction confirmations and reduced fees.

---

13. What is a state channel in Solidity?

Hide Answer

A state channel is a scalability solution for blockchain networks such as Ethereum, where Solidity is commonly used to write smart contracts. State channels provide a way for blockchain users to execute transactions and interact with smart contracts off-chain, improving scalability and efficiency while reducing fees.

A state channel functions by opening an off-chain channel between participants, through which multiple transactions, contractual interactions, or other off-chain computations can occur privately and securely. The initial state is locked on-chain with a smart contract, and the participants then update the state off-chain by signing transactions or agreements between them.

---

14. What is a cross-chain bridge in Solidity?

Hide Answer

A cross-chain bridge, although unrelated to Solidity as a language, is a solution in blockchain technology that enables communication and asset transfers between different blockchain networks, such as Ethereum (which uses Solidity for smart contracts) and Binance Smart Chain or other blockchains.

Cross-chain bridges work by connecting multiple blockchain platforms through the use of smart contracts, oracles, validators, and other components. These bridges facilitate the secure transfer of assets, including tokens and crypto-currencies, from one chain to another, often achieved by locking the original

15. What is a flash mint in Solidity?

Hide Answer

A flash mint is a specific feature available in some DeFi (Decentralized Finance) tokens, usually implemented using Solidity smart contracts. Flash minting allows users to mint a large amount of tokens instantly, with one critical condition - they must repay the minted tokens (plus any applicable fees) within the same block or transaction. This is done using a function built into the contract, such as flashMint().

Flash minting relies on the concept of flash loans, which enables users to borrow large amounts of tokens without providing collateral. The primary use case for flash mints includes arbitrage opportunities and executing complex financial interactions in DeFi platforms.

---

16. What do you know about smart contract wallet?

Hide Answer

A smart contract wallet is a type of cryptocurrency wallet that is built using smart contracts, commonly developed in languages like Solidity. It offers more advanced functionality and improved security compared to traditional wallets. These wallets operate on decentralized platforms, usually Ethereum, and allow users to store, manage, and interact with digital assets such as tokens, cryptocurrencies, and NFTs (Non-Fungible Tokens).

---

17. What is a multi-signature wallet in Solidity?

Hide Answer

A multi-signature (multisig) wallet in Solidity is a type of smart contract wallet that requires approval from multiple parties (or keys) before a transaction can be

The multisig wallet typically works by specifying a required number of approvals (known as the threshold) for any operation. For example, if a multisig wallet is set up with a 2-of-3 threshold, it means that at least two out of the three authorized users must sign and approve the transaction before it can be executed.

---

18. **How can you prevent overflow and underflow issues while working with uint in Solidity?**

Hide Answer

To prevent overflow and underflow issues, the SafeMath library provided by OpenZeppelin can be used. The library performs arithmetic operations with an added check for overflows and underflows, and will revert transactions if issues arise. Below is an example of its usage:

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/utils/math/SafeMath.sol";

contract MyContract {
    using SafeMath for uint256;

    function add(uint256 a, uint256 b) public pure returns (uint256) {
        return a.add(b);
    }

    function subtract(uint256 a, uint256 b) public pure returns (uint256) {
        return a.sub(b);
    }
}
```

---

19. **How can you implement a simple contract that accesses the block information in Solidity?**

```
pragma solidity ^0.8.0;

contract BlockInfo {

    function getBlockNumber() public view returns (uint256) {
        return block.number;
    }

    function getBlockTimestamp() public view returns (uint256) {
        return block.timestamp;
    }

    function getBlockDifficulty() public view returns (uint256) {
        return block.difficulty;
    }
}
```

This example demonstrates a simple contract that retrieves some of the most common block information: the current block number, the current block timestamp, and the block difficulty.

---

20. How can you implement a simple ERC20 token in Solidity?

Hide Answer

To create an ERC20 token, you must implement the ERC20 interface as defined in the EIP20 standard. As a developer, you can use the OpenZeppelin library, which provides tested and community-reviewed implementations of ERC20 tokens. Here's a simple example:

```
contract MyToken is ERC20 {

    constructor(uint256 initialSupply) ERC20("MyToken", "MTK") {
        _mint(msg.sender, initialSupply);
    }
}
```

In the example, the MyToken contract inherits the ERC20 contract from the OpenZeppelin library, which implements the required functions and behavior for an ERC20 token. The constructor sets the token's name, symbol, and mint the initial supply.

---

21. **How can you implement a time lock or delay on function execution in a Solidity contract?**

Hide Answer

A time lock or delay can be introduced by using block.timestamp to set a time limit before a function is executed. Here's a simple example:

```
    uint256 public unlockTime;
    address public owner;

    modifier onlyOwner {
        require(msg.sender == owner, "Not the owner");
        _;
    }

    modifier unlocked {
        require(block.timestamp >= unlockTime, "Function is locked");
        _;
    }

    constructor(uint256 _lockDuration) {
        owner = msg.sender;
        unlockTime = block.timestamp + _lockDuration;
    }

    function setUnlockTime(uint256 _lockDuration) public onlyOwner {
        unlockTime = block.timestamp + _lockDuration;
    }

    function executeRestrictedFunction() public onlyOwner unlocked {
        // Execute the restricted actions here...
    }
}
```

This example demonstrates a simple smart contract with a time lock on the executeRestrictedFunction function. The function can only be executed when the unlocked modifier's condition is met.

22. How can you implement an upgradable smart contract in Solidity?

Hide Answer

Upgradable contracts use a proxy pattern to separate the contract logic from the data storage, allowing developers to replace the contract logic while retaining the original data. The two most common patterns are the ‘unstructured storage’ and ‘eternal storage’ patterns.

A simple example using the ‘unstructured storage’ pattern:

```
contract Proxy {
    address public implementation;

    constructor(address _implementation) {
        implementation = _implementation;
    }

    fallback() external payable {
        address _implementation = implementation;
        assembly {
            let ptr := mload(0x40)
            calldatacopy(ptr, 0, calldatasize())
            let result := delegatecall(gas(), _implementation, ptr, calldatasize(), 0, 0)
            let size := returndatasize()
            returndatacopy(ptr, 0, size)

            switch result
            case 0 { revert(ptr, size) }
            default { return(ptr, size) }
        }
    }
}

// MyContract.sol
pragma solidity ^0.8.0;

import "./Proxy.sol";

contract MyContract is Proxy {
    uint256 public value;

    constructor(address _implementation) Proxy(_implementation) {}

    function setValue(uint256 _value) public {
        value = _value;
    }
}
```

This example demonstrates a basic upgradable contract using the Proxy contract to delegate calls to the implementation contract. Developers can replace the implementation without affecting the deployed MyContract data.

Keep in mind, this example is simplified and not recommended for production deployments. For a more robust upgradable contract system, refer to the OpenZeppelin library's proxy implementation, such as the TransparentUpgradeableProxy contract.

Function modifiers are a feature in Solidity that allows you to alter the behavior of a function. They can be used to add common requirements, such as access restrictions, preconditions, and state validation checks.

Example:

```
pragma solidity ^0.8.0;

contract ModifierExample {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    // Function modifier which restricts access to the owner
    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can call this function");
        _;
    }

    function restrictedFunction() public onlyOwner {
        // Execute function logic...
    }
}
```

In this example, the onlyOwner modifier restricts access to the restrictedFunction so that only the contract owner can execute it. The modifier checks the msg.sender and reverts the transaction if the sender is not the owner.

---

24. How can you implement a simple Owned contract pattern in Solidity?

Hide Answer

The Owned contract pattern uses an access control mechanism to restrict the access to certain functions to the contract creator. Below is an example:

```
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner, "Only the owner can call this function");
        _;
    }

    function changeOwner(address newOwner) public onlyOwner {
        owner = newOwner;
    }
}
```

---

25. Explain how you can send Ether to other addresses from within a smart contract.

Hide Answer

To send Ether from a smart contract, you can use the transfer, send, or call functions. The transfer and send functions are not recommended due to their limited gas stipend, which may cause issues with the receiving contracts. The call function is recommended for sending Ether. Here's an example:

```
pragma solidity ^0.8.0;

contract EtherSender {
    function sendEther(address payable recipient, uint256 amount) public payable {
        require(msg.value >= amount, "Insufficient Ether provided");
        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Ether transfer failed");
    }
}
```

This contract contains a sendEther function which takes a payable recipient address and an amount and transfers the specified amount of Ether to the recipient.

To implement an emergency stop, you can use a contract that sets a "stopped" state, and then use a function modifier to enforce that state. Below is an example:

```
pragma solidity ^0.8.0;

contract EmergencyStop {
    bool public stopped = false;
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    modifier stopInEmergency {
        require(!stopped, "Operation stopped due to an emergency");
        _;
    }

    modifier onlyOwner {
        require(msg.sender == owner
        require(msg.sender == owner, "Only the owner can call this function");
        _;
    }

    function toggleEmergencyStop() public onlyOwner {
        stopped = !stopped;
    }

    function restrictedFunction() public stopInEmergency {
        // Execute function logic...
    }
}
```

In this example, the EmergencyStop contract has a stopped state variable and a stopInEmergency modifier that restricts the execution of the restrictedFunction when the stopped state is true. The contract creator (owner) can toggle the emergency stop state using the toggleEmergencyStop function.

---

27. How can you access the balance of a contract?

Hide Answer

```
pragma solidity ^0.8.0;

contract BalanceFetcher {

    function getContractBalance() public view returns (uint256) {
        return address(this).balance;
    }

    // Fallback function to receive Ether
    receive() external payable {}
}
```

This example demonstrates a simple contract where you can retrieve the balance using the getContractBalance function. Using address(this).balance, you can get the current Ether balance of the contract.

---

28. **Explain how to implement simple access control tiering using function modifiers.**

Hide Answer

To implement an access control tiering system, define multiple privileged roles with distinct permissions within a contract. Then, use function modifiers to enforce role-based access to functions. Here's an example:

```
    address public admin;
    address public moderator;
    address public normalUser;

    constructor(address _moderator, address _normalUser) {
        admin = msg.sender;
        moderator = _moderator;
        normalUser = _normalUser;
    }

    modifier onlyAdmin() {
        require(msg.sender == admin, "Only admin can call this function");
        _;
    }

    modifier onlyModerator() {
        require(msg.sender == moderator, "Only moderator can call this function");
        _;
    }

    function adminFunction() public onlyAdmin {
        // Execute admin-only function logic...
    }

    function moderatorFunction() public onlyModerator {
        // Execute moderator-only function logic...
    }

    function userFunction() public {
        // Execute normal user function logic...
    }
}
```

In this example, the RoleBasedAccessControl contract uses the onlyAdmin and onlyModerator modifiers to restrict access to certain functions. The adminFunction can only be called by the admin, and the moderatorFunction can only be called by the moderator. The userFunction can be called by anyone.

---

29. Can a user send a transaction without being required to pay for gas? If yes, then how?

Hide Answer

In Ethereum, it's not possible for users to send transactions without paying for gas, as gas fees are required to compensate the miners for processing and validating transactions on the network. However, there are a few workarounds that can be

interact with contracts without ETH for gas fees. Instead, the gas fees are paid by a third party known as a "paymaster." A contract willing to accept "meta-transactions" must implement the GSN's relay recipient interface. When a user wants to execute a transaction, they create a "meta-transaction" and send it to a relay server instead of the Ethereum network directly. The relay server forwards the transaction to the contract while paying the gas fee. The paymaster (or the contract itself) will later compensate the relay server.

Flash bots: Flash bots can help users have their transaction included in a block without paying a direct gas fee. To achieve this, the user includes the payment in the transaction and sends it to a flash bot-enabled miner. The miner benefits from the payment while also circumventing the regular "first-price auction" model for gas. However, this approach is complex and primarily used for more advanced use cases such as arbitrage or liquidations.

EIP-3074 (Authenticator): EIP-3074 is a proposed protocol upgrade that introduces a new opcode (AUTH) allowing users to delegate transaction processing to a third party (known as an "Authenticator") who will pay the gas fees. Users sign a message that permits specific operations, and the Authenticator creates and sends the transaction, paying the fees themselves. Once EIP-3074 is implemented, this approach can be a viable solution for enabling users to send transactions without directly paying gas fees.

Looking for remote developer job at US companies?

Work at Fortune 500 companies and fast-scaling startups from the comfort of your home

Apply Now

WRAPPING UP

We hope the above list of Solidity interview questions will give you a headstart for cracking your ideal Solidity developer job. By dividing the questions based on the level of difficulty we have tried to provide you an overview of the important concepts that define Solidity.

If you want to work as a Solidity developer for some of Silicon Valley's most outstanding organizations, take the Turing test today to be considered for these positions. Leave a

About us

Work at Fortune 500 companies and fast-scaling startups from the comfort of your home

Apply now

Apply for the latest remote jobs

Full-Stack Developer

FULL-TIME REMOTE JOB

Posted 4 days ago 1-10 Edtech

Job description templates →

Learn how to write a clear and comprehensive job description to attract highly skilled Solidity developers to your organization.

Solidity developer resume tips →

Turing.com lists out the do's and don'ts behind a great resume to help you find a top remote Solidity developer job.

Check out more interview questions

React Node.js Python AWS JavaScript
HTML Java Flutter C# SQL Angular
iOS iOS PHP PHP

+ See more skills

Based on your role

Front-end Full Stack Backend Machine Learning
Data Science AI Cloud DevOps
Remote Developer Software Engineer

+ See more roles

Hire remote developers

Tell us the skills you need and we'll find the best developer for you in days, not weeks.

Hire Developers

About us

Engineering services

LLM training and enhancement

Generative AI

AI/ML

Custom engineering

All services →

On-demand talent

Technical professionals

Development teams

For developers

Browse remote jobs

Get hired

Developer reviews

Developer resources

Tech interview questions

Resources

Blog

More resources →

Company

About us

Connect

Contact us

Help center

Sitemap

Terms of service

Privacy policy

Privacy settings

1900 Embarcadero Road Palo Alto, CA, 94303

© 2024 Turing