Building DeFi – Quiz

Cryptoeconomics

Question 1

What does the constant product formula `x * y = k` guarantee in an AMM?

A. The price of each token is always equal to 1 USD
B. The product of the two token reserves stays constant after every swap
C. The total number of tokens in the pool never changes
D. The pool always contains equal USD value of both tokens

Question 1

What does the constant product formula `x * y = k` guarantee in an AMM?

A. The price of each token is always equal to 1 USD

B. The product of the two token reserves stays constant after every swap

C. The total number of tokens in the pool never changes

D. The pool always contains equal USD value of both tokens

Answer: B

In an AMM using $x \cdot y = k$, whenever a trader swaps token A for token B, the amounts change but their product k remains constant. This mathematical constraint automatically sets the exchange rate without an order book.

Question 2

If a DEX pool has 100 tokenA (x) and 100 tokenB (y), giving $k = 10{,}000$, and a trader swaps in 10 tokenA, approximately how much tokenB do they receive?

A. 10 tokenB (1:1 rate)

B. Approximately 9.09 tokenB

C. Exactly 10.1 tokenB

D. 100 tokenB

Question 2

If a DEX pool has 100 tokenA (x) and 100 tokenB (y), giving $k = 10{,}000$, and a trader swaps in 10 tokenA, approximately how much tokenB do they receive?

A. 10 tokenB (1:1 rate)

B. Approximately 9.09 tokenB

C. Exactly 10.1 tokenB

D. 100 tokenB

Answer: B

Using `amountOut = (reserveOut * amountIn) / (reserveIn + amountIn)`: $(100 \times 10)/(100 + 10) = 1000/110 \approx 9.09$. The trader receives less than 10 because buying tokenB increases its price (price impact).

Question 3

What is "slippage" in the context of a DEX swap?

A. A fee charged by the Ethereum network for transactions
B. The difference between the expected price and the actual execution price
C. The time delay between submitting and confirming a swap
D. A penalty for removing liquidity too early

Question 3

What is “slippage” in the context of a DEX swap?

A. A fee charged by the Ethereum network for transactions

B. The difference between the expected price and the actual execution price

C. The time delay between submitting and confirming a swap

D. A penalty for removing liquidity too early

Answer: B

Slippage occurs because the act of swapping changes the pool’s reserves, moving the price. Large trades cause more slippage. The `require(amountOut >= _minOut, "Slippage too high")` check protects users from getting far less than expected.

Question 4

What does it mean to "add liquidity" to a DEX pool?

A. Borrowing tokens from the protocol
B. Depositing both tokens into the pool's reserves to enable trading
C. Converting one token to the other at the current price
D. Removing your tokens and receiving a fixed interest payment

Question 4

What does it mean to "add liquidity" to a DEX pool?

A. Borrowing tokens from the protocol
B. Depositing both tokens into the pool's reserves to enable trading
C. Converting one token to the other at the current price
D. Removing your tokens and receiving a fixed interest payment

Answer: B
Liquidity providers deposit proportional amounts of both tokens, increasing the pool's reserves. In return they receive LP tokens representing their share. Their deposits are used to fill traders' swaps.

Question 5

Why do AMM-based DEXes not need a traditional order book?

A. They use off-chain matching engines instead
B. The constant product formula automatically determines the price from reserve ratios
C. All trades are executed at a fixed government-set price
D. They only allow trades between whitelisted addresses

Question 5

Why do AMM-based DEXes not need a traditional order book?

A. They use off-chain matching engines instead

B. The constant product formula automatically determines the price from reserve ratios

C. All trades are executed at a fixed government-set price

D. They only allow trades between whitelisted addresses

Answer: B

Traditional exchanges match buy and sell orders in an order book. AMMs replace this with a mathematical formula: the price at any moment is simply reserveB/reserveA. Trades execute automatically against the pool's reserves.

Question 6

How does a DEX's `addLiquidity` function obtain tokens from the liquidity provider?

A. It mints new tokens directly

B. It calls `transferFrom` on each ERC-20 token contract after the user has called `approve`

C. The user must first send tokens to the DEX contract's address manually

D. It borrows tokens from an oracle

Question 6

How does a DEX's `addLiquidity` function obtain tokens from the liquidity provider?

A. It mints new tokens directly

B. It calls `transferFrom` on each ERC-20 token contract after the user has called `approve`

C. The user must first send tokens to the DEX contract's address manually

D. It borrows tokens from an oracle

Answer: B

The DEX uses the ERC-20 approve+transferFrom pattern from L10: the liquidity provider first calls `approve(dexAddress, amount)` on each token, then the DEX's `addLiquidity` function calls `tokenA.transferFrom(msg.sender, address(this), _amountA)` to pull the tokens in.

Question 7

As a trader swaps more and more tokenA for tokenB from the same pool, what happens to the price of tokenA?

A. It stays constant because k is fixed
B. It gradually decreases (tokenA becomes cheaper)
C. It gradually increases because the pool has more tokenA and less tokenB
D. It doubles with each swap

Question 7

As a trader swaps more and more tokenA for tokenB from the same pool, what happens to the price of tokenA?

A. It stays constant because k is fixed

B. It gradually decreases (tokenA becomes cheaper)

C. It gradually increases because the pool has more tokenA and less tokenB

D. It doubles with each swap

Answer: C

Each swap adds tokenA to the pool and removes tokenB. As reserveA grows and reserveB shrinks, the ratio reserveB/reserveA falls, meaning tokenA is worth less tokenB — its relative price drops. This is the price impact shown on the AMM curve.

Question 8

What is an `interface` in Solidity?

A. A contract that stores token balances

B. A declaration of function signatures without implementation, used to call other contracts

C. A modifier that controls access to functions

D. A mapping between function names and their gas costs

Question 8

What is an `interface` in Solidity?

A. A contract that stores token balances

B. A declaration of function signatures without implementation, used to call other contracts

C. A modifier that controls access to functions

D. A mapping between function names and their gas costs

Answer: B

An interface (e.g., `interface IERC20`) declares what functions a contract has and their signatures, but contains no code. Your contract can then call those functions on any address that implements the interface, enabling cross-contract calls.

Question 9

What is the `external` visibility keyword used for in Solidity?

A. Functions that can only be called from within the same contract
B. Functions that can only be called from outside the contract (not internally)
C. Functions that automatically emit events
D. Functions that cost zero gas to call

Question 9

What is the `external` visibility keyword used for in Solidity?

A. Functions that can only be called from within the same contract

B. Functions that can only be called from outside the contract (not internally)

C. Functions that automatically emit events

D. Functions that cost zero gas to call

Answer: B

`external` functions can only be called from outside the contract — other contracts or transactions. They cannot be called internally with `this.functionName()`. Interface functions are always `external` because they describe how to call the contract from the outside.

Question 10

In the SimpleDEX contract, why does the swap function need to know which token is being sent in (`_tokenIn`)?

A. To charge a different fee for each token

B. To determine which reserve is the "in" reserve and which is the "out" reserve for the price calculation

C. To check if the token is on a whitelist

D. To emit the correct event name

Question 10

In the SimpleDEX contract, why does the swap function need to know which token is being sent in (`_tokenIn`)?

A. To charge a different fee for each token

B. To determine which reserve is the "in" reserve and which is the "out" reserve for the price calculation

C. To check if the token is on a whitelist

D. To emit the correct event name

Answer: B

The swap function must determine: if `_tokenIn == tokenA`, then reserveIn = reserveA and reserveOut = reserveB (and vice versa). This directionality is needed to correctly apply `amountOut = (reserveOut * amountIn) / (reserveIn + amountIn)`.

Question 11

After a swap in the SimpleDEX, what must the contract do to its stored reserve values?

A. Reset both reserves to zero
B. Update reserveIn (increase by amountIn) and reserveOut (decrease by amountOut)
C. Leave reserves unchanged since the tokens are already transferred
D. Double the reserves to prepare for the next swap

Question 11

After a swap in the SimpleDEX, what must the contract do to its stored reserve values?

A. Reset both reserves to zero

B. Update reserveIn (increase by amountIn) and reserveOut (decrease by amountOut)

C. Leave reserves unchanged since the tokens are already transferred

D. Double the reserves to prepare for the next swap

Answer: B

After transferring tokens, the contract must update its reserve accounting: `reserveIn += amountIn` and `reserveOut -= amountOut`. These stored values are used by the next swap to calculate prices. Without updating them, the price formula would use stale data.

Question 12

What is the purpose of LP (Liquidity Provider) tokens?

A. They are used as currency within the DEX to pay trading fees
B. They represent a proportional ownership share of the pool, redeemable for the underlying tokens
C. They grant voting rights over the DEX protocol's parameters
D. They are burned every time a swap occurs

Question 12

What is the purpose of LP (Liquidity Provider) tokens?

A. They are used as currency within the DEX to pay trading fees

B. They represent a proportional ownership share of the pool, redeemable for the underlying tokens

C. They grant voting rights over the DEX protocol's parameters

D. They are burned every time a swap occurs

Answer: B

LP tokens are minted when liquidity is added and burned when it is withdrawn. If you hold 10% of all LP tokens, you can redeem them for 10% of the pool's current reserves. They also accumulate trading fees over time.

Question 13

In the DEX walkthrough, what observable effect happens to the reserves after multiple swaps of tokenA for tokenB?

A. Both reserves remain equal at all times
B. ReserveA increases and reserveB decreases, making tokenB relatively more expensive
C. Both reserves shrink as tokens are consumed
D. The pool rebalances automatically back to 50/50

Question 13

In the DEX walkthrough, what observable effect happens to the reserves after multiple swaps of tokenA for tokenB?

A. Both reserves remain equal at all times

B. ReserveA increases and reserveB decreases, making tokenB relatively more expensive

C. Both reserves shrink as tokens are consumed

D. The pool rebalances automatically back to 50/50

Answer: B

Each swap of tokenA for tokenB adds tokenA to the pool and removes tokenB. After many such swaps, reserveA is large and reserveB is small. The price of tokenB (in terms of tokenA) rises significantly, demonstrating price impact.

Question 14

What is an escrow contract?

A. A contract that automatically trades tokens on a schedule
B. A contract that holds funds in trust until a condition is met, then releases them
C. A contract that generates interest on deposited tokens
D. A contract that verifies the identity of participants

Question 14

What is an escrow contract?

A. A contract that automatically trades tokens on a schedule

B. A contract that holds funds in trust until a condition is met, then releases them

C. A contract that generates interest on deposited tokens

D. A contract that verifies the identity of participants

Answer: B

An escrow contract acts as a neutral intermediary: the buyer deposits funds, the seller delivers goods/services, and the contract releases payment upon confirmation. If there is a dispute, an arbiter can redirect funds.

Question 15

What is an `enum` in Solidity?

A. A function that enumerates all addresses in a mapping
B. A user-defined type with a fixed set of named constant values
C. A loop that iterates a fixed number of times
D. An array whose length cannot change

Question 15

What is an `enum` in Solidity?

A. A function that enumerates all addresses in a mapping

B. A user-defined type with a fixed set of named constant values

C. A loop that iterates a fixed number of times

D. An array whose length cannot change

Answer: B

`enum State { AWAITING_PAYMENT, AWAITING_DELIVERY, COMPLETE, REFUNDED }` defines four named states. Under the hood, Solidity stores them as uint8 (0, 1, 2, 3). Enums make code readable and prevent invalid state values.

Question 16

What does `payable` mean when applied to a function?

A. The function costs extra gas to call
B. The function can receive ETH along with the transaction
C. The function automatically sends ETH to the caller
D. The function is only callable by token holders

Question 16

What does `payable` mean when applied to a function?

A. The function costs extra gas to call

B. The function can receive ETH along with the transaction

C. The function automatically sends ETH to the caller

D. The function is only callable by token holders

Answer: B

A `payable` function accepts ETH sent with the transaction. Without this keyword, any attempt to send ETH to the function will revert. The amount sent is accessible as `msg.value`.

Question 17

What does `msg.value` contain inside a `payable` function?

A. The total ETH balance of the contract
B. The caller's wallet ETH balance
C. The amount of ETH (in wei) sent with the current transaction
D. The gas price of the current transaction

Question 17

What does `msg.value` contain inside a `payable` function?

A. The total ETH balance of the contract

B. The caller's wallet ETH balance

C. The amount of ETH (in wei) sent with the current transaction

D. The gas price of the current transaction

Answer: C

`msg.value` is the amount of ETH (measured in wei, the smallest unit) sent along with the transaction. In the escrow deposit function, `require(msg.value == amount)` verifies the buyer sent exactly the agreed amount.

Question 18

In the escrow's `confirmDelivery` function, `payable(seller).transfer(address(this).balance)` does what?

A. Creates a new ERC-20 token for the seller
B. Sends all ETH held by the escrow contract to the seller's address
C. Transfers LP tokens to the seller
D. Locks the seller's funds for 30 days

Question 18

In the escrow's `confirmDelivery` function, `payable(seller).transfer(address(this).balance)` does what?

A. Creates a new ERC-20 token for the seller

B. Sends all ETH held by the escrow contract to the seller's address

C. Transfers LP tokens to the seller

D. Locks the seller's funds for 30 days

Answer: B

`address(this).balance` is the total ETH held by the contract. Calling `.transfer(amount)` on a `payable` address sends that ETH to the recipient. Together, this line releases all escrowed funds to the seller upon delivery confirmation.

Question 19

What is `block.timestamp` used for in the escrow time-lock?

A. The Unix timestamp of when the contract was deployed
B. The current block's timestamp, used to enforce deadlines
C. The time the buyer submitted their transaction
D. The interval between consecutive Ethereum blocks

Question 19

What is `block.timestamp` used for in the escrow time-lock?

A. The Unix timestamp of when the contract was deployed

B. The current block's timestamp, used to enforce deadlines

C. The time the buyer submitted their transaction

D. The interval between consecutive Ethereum blocks

Answer: B

`block.timestamp` is the timestamp of the current block in Unix seconds. In the escrow, `require(block.timestamp < deadline)` prevents deposits after the deadline, and enables automatic refunds if the seller does not deliver in time.

Question 20

In the escrow contract, who is the `arbiter` and what can they do?

A. The contract deployer, who can change the token being escrowed
B. A trusted third party who can resolve disputes by redirecting funds to buyer or seller
C. An automated oracle that checks delivery status
D. The Ethereum Foundation, which approves all escrow contracts

Question 20

In the escrow contract, who is the `arbiter` and what can they do?

A. The contract deployer, who can change the token being escrowed

B. A trusted third party who can resolve disputes by redirecting funds to buyer or seller

C. An automated oracle that checks delivery status

D. The Ethereum Foundation, which approves all escrow contracts

Answer: B

The arbiter is a pre-agreed neutral party (set at deployment). If a dispute arises, the arbiter can call the `dispute()` function to refund the buyer. This prevents deadlock situations where neither party can access the funds.

Question 21

What game theory concept explains why the escrow contract creates correct incentives for both buyer and seller?

A. The prisoner's dilemma
B. Commitment device — locking funds makes defection costly
C. Dominant strategy equilibrium
D. The tragedy of the commons

Question 21

What game theory concept explains why the escrow contract creates correct incentives for both buyer and seller?

A. The prisoner's dilemma

B. Commitment device — locking funds makes defection costly

C. Dominant strategy equilibrium

D. The tragedy of the commons

Answer: B

The escrow acts as a commitment device (from L07): the buyer commits funds before delivery, making it credible that they will pay. The seller sees the locked funds and is incentivized to deliver. Both parties' rational strategies align to complete the trade.

Question 22

What is the reentrancy vulnerability, and which pattern prevents it?

A. Calling the same function twice; prevented by making functions `pure`

B. A malicious contract calling back into your contract before state is updated; prevented by Checks-Effects-Interactions

C. Integer overflow when adding large numbers; prevented by using uint256

D. Front-running transactions; prevented by using private functions

Question 22

What is the reentrancy vulnerability, and which pattern prevents it?

A. Calling the same function twice; prevented by making functions `pure`

B. A malicious contract calling back into your contract before state is updated; prevented by Checks-Effects-Interactions

C. Integer overflow when adding large numbers; prevented by using uint256

D. Front-running transactions; prevented by using private functions

Answer: B

In a reentrancy attack, a malicious contract's fallback function re-enters your contract before you update balances. The CEI pattern fixes this: (1) Check conditions, (2) update Effects (state), (3) then Interact (call external contracts). Never call external contracts before updating state.

Question 23

In the "bad" reentrancy example, why is sending ETH BEFORE updating the balance dangerous?

A. It wastes gas by doing unnecessary work
B. The attacker's fallback function can re-enter and withdraw again before the balance is set to zero
C. Solidity reverts any transaction that sends ETH before updating mappings
D. It triggers an integer overflow in the balance variable

Question 23

In the "bad" reentrancy example, why is sending ETH BEFORE updating the balance dangerous?

A. It wastes gas by doing unnecessary work

B. The attacker's fallback function can re-enter and withdraw again before the balance is set to zero

C. Solidity reverts any transaction that sends ETH before updating mappings

D. It triggers an integer overflow in the balance variable

Answer: B

If you call `transfer()` before setting `balance[msg.sender] = 0`, the attacker's fallback function fires, which calls your `withdraw()` again. Your contract still shows the old balance, so it sends ETH again. This repeats until your contract is drained. The 2016 DAO hack used this exact exploit.

Question 24

What does `composability` mean in the context of DeFi smart contracts?

A. All DeFi contracts must be written by the same team
B. Contracts can call and build on each other like building blocks, creating complex financial protocols
C. Token balances can be combined into a single account
D. Gas fees are shared between contracts that interact

Question 24

What does `composability` mean in the context of DeFi smart contracts?

A. All DeFi contracts must be written by the same team

B. Contracts can call and build on each other like building blocks, creating complex financial protocols

C. Token balances can be combined into a single account

D. Gas fees are shared between contracts that interact

Answer: B

Composability ("money legos") means that contracts interact via standard interfaces. A lending protocol can use your ERC-20 token; a yield aggregator can use the lending protocol. Each layer builds on the layer below, enabling complex DeFi without any central coordinator.

Question 25

In Remix's unit testing plugin, what is the purpose of writing a test that calls `transfer()` and then asserts the recipient's balance?

A. To measure the gas cost of the transfer
B. To automatically deploy the contract to mainnet
C. To verify the function behaves correctly and catch regressions if the code changes
D. To generate the ABI for the contract

Question 25

In Remix's unit testing plugin, what is the purpose of writing a test that calls `transfer()` and then asserts the recipient's balance?

A. To measure the gas cost of the transfer

B. To automatically deploy the contract to mainnet

C. To verify the function behaves correctly and catch regressions if the code changes

D. To generate the ABI for the contract

Answer: C

Unit tests assert that a function produces the correct output for given inputs. If you later modify `transfer()` and accidentally break it, the test will fail and alert you. Good test coverage prevents deploying buggy contracts to mainnet where they cannot be changed.

Question 26

Which item is on the security checklist before deploying any smart contract?

A. The contract must be written in Python first, then translated to Solidity
B. All state-changing functions must emit at least one event
C. The contract must have at least 1,000 lines of code
D. The constructor must set msg.value to zero

Question 26

Which item is on the security checklist before deploying any smart contract?

A. The contract must be written in Python first, then translated to Solidity

B. All state-changing functions must emit at least one event

C. The contract must have at least 1,000 lines of code

D. The constructor must set msg.value to zero

Answer: B

Emitting events on every state change is a security and auditability best practice. Events create an immutable audit trail in the transaction log, allowing off-chain monitoring tools to detect suspicious activity and enabling block explorers to show contract history.

Question 27

What makes the DEX contract vulnerable to front-running if there is no slippage protection?

A. Without slippage protection, the DEX charges no fees
B. A miner/validator can insert their own swap before yours, worsening your price, and you have no recourse
C. The contract's reserves can go negative without a minimum check
D. Other users can cancel your transaction

Question 27

What makes the DEX contract vulnerable to front-running if there is no slippage protection?

A. Without slippage protection, the DEX charges no fees

B. A miner/validator can insert their own swap before yours, worsening your price, and you have no recourse

C. The contract's reserves can go negative without a minimum check

D. Other users can cancel your transaction

Answer: B

MEV (Maximal Extractable Value) bots see pending transactions and insert a swap before yours ("sandwich attack"). Without a `require(amountOut >= _minOut)` guard, you must accept whatever the contract gives you — even near zero. The minOut parameter lets you reject unfavorable executions.

Question 28

How does the SimpleDEX price calculation formula `(reserveOut * amountIn) / (reserveIn + amountIn)` differ from a simple ratio?

A. It adds a 0.3% fee to the calculation automatically

B. It accounts for the price impact of the trade itself by adding amountIn to reserveIn in the denominator

C. It divides by zero when reserveIn equals amountIn

D. It uses floating-point arithmetic to get exact results

Question 28

How does the SimpleDEX price calculation formula `(reserveOut * amountIn) / (reserveIn + amountIn)` differ from a simple ratio?

A. It adds a 0.3% fee to the calculation automatically

B. It accounts for the price impact of the trade itself by adding amountIn to reserveIn in the denominator

C. It divides by zero when reserveIn equals amountIn

D. It uses floating-point arithmetic to get exact results

Answer: B

A naive price would be reserveOut/reserveIn $\times$ amountIn. The correct formula adds amountIn to the denominator, accounting for the fact that by the time the swap executes, the "in" reserve has grown by amountIn. This correctly implements the constant product invariant.

Question 29

What does `address(this).balance` return in a Solidity contract?

A. The deployer's ETH balance
B. The total ETH held by the contract itself
C. The amount of ETH sent in the current transaction
D. The gas remaining in the current execution

Question 29

What does `address(this).balance` return in a Solidity contract?

A. The deployer's ETH balance

B. The total ETH held by the contract itself

C. The amount of ETH sent in the current transaction

D. The gas remaining in the current execution

Answer: B

`this` refers to the current contract. `address(this)` casts it to an address. `.balance` is a property of any address giving its ETH balance in wei. In the escrow, this is the buyer's deposited ETH waiting to be released.

Question 30

In the escrow, why is a `deadline` (`block.timestamp` check) important even when an arbiter exists?

A. Without a deadline the arbiter's address could be set to the zero address

B. If both parties disappear or the arbiter goes offline, the buyer's funds would be locked forever

C. `block.timestamp` is required by the ERC-20 standard for escrow contracts

D. The deadline limits how many swaps the DEX can execute

Question 30

In the escrow, why is a `deadline` (`block.timestamp` check) important even when an arbiter exists?

A. Without a deadline the arbiter's address could be set to the zero address

B. If both parties disappear or the arbiter goes offline, the buyer's funds would be locked forever

C. `block.timestamp` is required by the ERC-20 standard for escrow contracts

D. The deadline limits how many swaps the DEX can execute

Answer: B

Without a time-lock, if the seller never delivers and the arbiter never acts (offline, corrupted, or colluding), the buyer's ETH would be permanently locked in the contract. The deadline enables the buyer (or anyone) to claim a refund after the deadline passes.

Question 31

What must a developer do BEFORE the SimpleDEX's `addLiquidity` function can call `tokenA.transferFrom(msg.sender, ...)`?

A. Deploy tokenA after the DEX so it knows the DEX address

B. The liquidity provider must first call `tokenA.approve(dexAddress, amount)`

C. The DEX must own some tokenA already

D. The liquidity provider must send ETH to the DEX first

Question 31

What must a developer do BEFORE the SimpleDEX's `addLiquidity` function can call `tokenA.transferFrom(msg.sender, ...)`?

A. Deploy tokenA after the DEX so it knows the DEX address

B. The liquidity provider must first call `tokenA.approve(dexAddress, amount)`

C. The DEX must own some tokenA already

D. The liquidity provider must send ETH to the DEX first

Answer: B

ERC-20's `transferFrom` requires prior authorization via `approve`. If Alice wants to add liquidity, she must call `tokenA.approve(dexAddress, amountA)` first. Without this, the DEX's `transferFrom` call will revert with "insufficient allowance".

Question 32

What is the key difference between how the escrow contract handles ETH vs how the DEX handles tokens?

A. The escrow uses mapping balances; the DEX uses actual contract balances

B. ETH is received via `payable` functions and `msg.value`; ERC-20 tokens require the approve+transferFrom pattern

C. The DEX uses ETH natively; the escrow uses ERC-20 tokens only

D. Both use identical mechanisms — ETH and ERC-20 work the same way in Solidity

Question 32

What is the key difference between how the escrow contract handles ETH vs how the DEX handles tokens?

A. The escrow uses mapping balances; the DEX uses actual contract balances

B. ETH is received via `payable` functions and `msg.value`; ERC-20 tokens require the approve+transferFrom pattern

C. The DEX uses ETH natively; the escrow uses ERC-20 tokens only

D. Both use identical mechanisms — ETH and ERC-20 work the same way in Solidity

Answer: B

ETH has first-class support: mark the function `payable` and ETH arrives automatically in `msg.value`, no approval needed. ERC-20 tokens are separate contracts — to move them you must use the approve+transferFrom pattern because the token contract enforces its own rules.

Question 33

What does the "Money Legos" metaphor describe about DeFi?

A. DeFi protocols physically resemble Lego bricks when drawn as diagrams
B. DeFi protocols snap together via standard interfaces to build complex financial services from simple parts
C. All DeFi contracts must be deployed in the same transaction
D. DeFi is only available to large financial institutions, like building blocks of the financial system

Question 33

What does the "Money Legos" metaphor describe about DeFi?

A. DeFi protocols physically resemble Lego bricks when drawn as diagrams

B. DeFi protocols snap together via standard interfaces to build complex financial services from simple parts

C. All DeFi contracts must be deployed in the same transaction

D. DeFi is only available to large financial institutions, like building blocks of the financial system

Answer: B

Like Lego bricks that connect via a standard interface, DeFi contracts connect via ERC-20 and other standards. Uniswap uses ERC-20 tokens; Aave accepts Uniswap LP tokens as collateral; aggregators route through both. Each "lego" is usable independently or as part of a larger structure.

Question 34

What does a production-grade DeFi contract need that our SimpleDEX does not have?

A. A `pragma solidity` statement
B. Audits, reentrancy guards, and formal verification before mainnet deployment
C. More state variables
D. Deployment on Remix instead of Hardhat

Question 34

What does a production-grade DeFi contract need that our SimpleDEX does not have?

A. A `pragma solidity` statement

B. Audits, reentrancy guards, and formal verification before mainnet deployment

C. More state variables

D. Deployment on Remix instead of Hardhat

Answer: B

Production DeFi contracts managing real funds require professional security audits (e.g., by Trail of Bits, OpenZeppelin), formal verification of critical invariants, reentrancy guards, and extensive testing. Our SimpleDEX is a learning tool, not production-ready code.

Question 35

In Solidity 0.8.x, what happens automatically if an arithmetic operation would overflow a `uint256`?

A. The value wraps around to zero (same as pre-0.8.x behavior)

B. The transaction reverts with an overflow error

C. The value is capped at $2^{256} - 1$

D. A warning event is emitted but execution continues

Question 35

In Solidity 0.8.x, what happens automatically if an arithmetic operation would overflow a `uint256`?

A. The value wraps around to zero (same as pre-0.8.x behavior)

B. The transaction reverts with an overflow error

C. The value is capped at $2^{256} - 1$

D. A warning event is emitted but execution continues

Answer: B

Since Solidity 0.8.0, integer arithmetic has built-in overflow/underflow protection. Any operation that would exceed the type's range automatically reverts. This eliminates a whole class of pre-0.8 bugs that required SafeMath libraries.

Question 36

Which combination of concepts from L06, L07, and L10 does the SimpleDEX contract directly apply?

A. Hash functions (L03), consensus (L04), and tokenomics (L05)

B. AMM constant product formula (L06), smart contract interactions (L07), and ERC-20 approve+transferFrom (L10)

C. Blockchain structure (L02), regulation (L08), and voting systems (L09)

D. Cryptographic proofs (L03), gas optimization (L07), and structs (L10)

Question 36

Which combination of concepts from L06, L07, and L10 does the SimpleDEX contract directly apply?

A. Hash functions (L03), consensus (L04), and tokenomics (L05)

B. AMM constant product formula (L06), smart contract interactions (L07), and ERC-20 approve+transferFrom (L10)

C. Blockchain structure (L02), regulation (L08), and voting systems (L09)

D. Cryptographic proofs (L03), gas optimization (L07), and structs (L10)

Answer: B

The SimpleDEX combines three prior lessons: the $x \cdot y = k$ AMM formula from L06, smart contract cross-contract calls and interface patterns from L07, and the ERC-20 approve+transferFrom token mechanic from L10. L11 is the implementation layer where all three converge.