EVM gas costs:

* **EVM Opcodes:** [evm.codes](https://www.evm.codes/" \t "/home/direb/Documents\\x/_blank) is a valuable interactive resource for understanding EVM opcodes.
* **Deterministic Gas Costs:** Gas costs are deterministically calculated based on the opcodes used in a transaction.
* **Fixed vs. Variable Gas:** Simple opcodes (e.g., ADD, MUL) have fixed gas costs. Complex opcodes (e.g., SSTORE) have gas costs that depend on several factors.
* **Gas Refunds:** Gas refunds exist for certain operations, such as clearing a storage slot.
* Think of Ethereum having two types of accounts: regular people's accounts (EOAs) and smart contract accounts. The system treats them pretty much the same.
* Every account has a public address (like an email) and some money (balance). Smart contract accounts also store the contract's code and data.
* When you (a regular account) call a smart contract, the system needs to know who's calling, how much money they're sending, and what function in the contract they want to use.
* Solidity (the programming language) makes it easy to connect your call to the right function in the contract and gives you access to info like who's calling (msg.sender) and how much money they sent (msg.value).

**Storage Variables**

* variables declared in contract scope are storage variables.
* solidity store those variables in contiguous storage slot.

**Function**

* · Functions allow you to read and modify data within a smart contract.
* Declaring a state variable public automatically creates a getter function (e.g., x() in the example) to read its value.You can define custom functions (like changeX()) to change the values of state variables.

**Message calls**

**.**send value and calldata to contract.

.The first message call is the beginning of the transaction(EOA->contract)

.each subsequent message call is part of the same transaction(contract->contract).

.The trnsaction and any state changes only complete when the intial function call finishes execution.

**Message Call Breakdown**

* as we saw message calls can contain gas.value and calldata
* These message vlues become available as global in solidity
* msg.sender-who made the last message call
* asg.value-amount in wei sent
* asg.data-calldata
* asg.sig-the function identifier.

**Revert**

* we talked to a contract with message calles.
* A contract can REVERT a call,negating all state changes.
* each calling contract can choose to handle that success ,REVERT as well.

**Reverting:** Stops execution, undoes changes, refunds gas.

**Solidity (>=0.8.0) Error Handling:**

* + assert(): Internal errors (code bug), uses all gas.
  + require(): User errors (invalid input), refunds gas.
  + revert(): General errors, refunds gas, custom messages.
* **Calldata**
* Data sent with a transaction to a smart contract.
* **Function Call Encoding:**
  + Hash function signature (e.g., approve(uint256)) with keccak256.
  + Take the first 4 bytes of the hash (function selector).
  + Encode function arguments, padded to 32 bytes (256 bits/64 hex characters).
  + Concatenate the function selector and encoded arguments.
* **Example:** Calling approve(15):
  + Selector: 0xb759f954 (first 4 bytes of keccak256("approve(uint256)"))
  + Argument (15): 0x000000000000000000000000000000000000000000000000000000000000000f
  + Calldata: 0xb759f954000000000000000000000000000000000000000000000000000000000000000f
* **Applies to:** Both EOA transactions and contract-to-contract calls.

## Escrow

Escrow is an agreement for transferring funds in exchange for goods/services.

* Funds held by a third party until a condition (service/good provided) is met.
* A third party can "arbitrate" or approve the transfer.

· **Use Cases:** Real estate, charities, marketplaces.

· **Significance:** Fundamental smart contract application, easy to implement, powerful.

· **Goal:** To use escrows to launch new decentralized applications.

* **Solidity Arrays**
* **Data Location:** Important to understand where the array is stored (memory, storage, calldata).
* **Memory Allocation:** Need to know if the array has a fixed size or can be resized dynamically.

**Structs**

* **-**group variables under a single name.
* **-**can be stored in a different data locations.
* **-**can go within others struct/array/mapping.

**Mapping**

**Mapping** is A data type that maps values of one type to another (e.g., address to uint).One of the most important data types in Solidity.

* **-**key/value hash lookup.
* **-**storage only.
* - cannot be passed as the argument.

**Use Case**: Efficiently track key-value pairs, like address to balance.

**Efficiency**: Faster lookups than iterating through an array of structs. Mappings use hashing for direct access.

**Syntax**: mapping(keyType => valueType) mappingName; (e.g., mapping(address => uint) balances;)

## Voting Smart Contract

* any memeber can make a proposal
* memebers can vote on whether theey like it.
* after a yes threshold is met,the proposal is excuted.
* the proposal contains calldate to be sent to contract.
* since the proposal can execute a message call,it could be anything from moving some funds to buying an NFT!.the contract become like an EOA that requires voter approval for each of its actions.
* This voting contract allows members to create and vote on proposals. Each proposal tracks its votes and, upon passing, executes by sending calldata to a target smart contract. This could be used for various purposes, such as upgrading a protocol by sending the upgrade(address) function signature and the new implementation address as calldata.