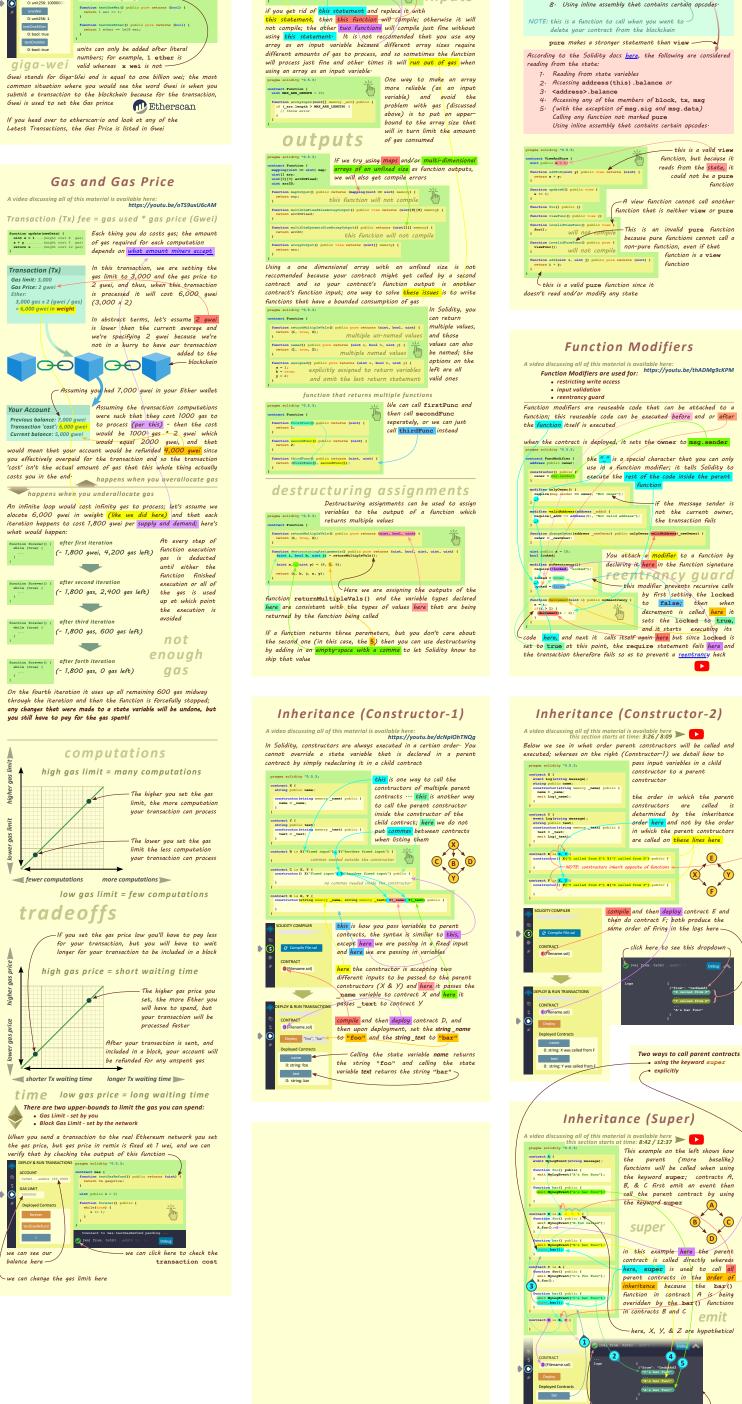


in this file explorer —

contract SimpleStorage {
 string public text;

function set(string memory _text) pub text = _text;

variables; we do this by using the



Invalid Functions

dity, there are certain data types that cannot be input

oderv2; allowed and "[]" is not reccomended.

View and Pure Functions

view functions do not modify the state of the blockchain

pure functions do not modify the state of the blockchain, nor do they read the state either

According to the Solidity docs here, The following statements are

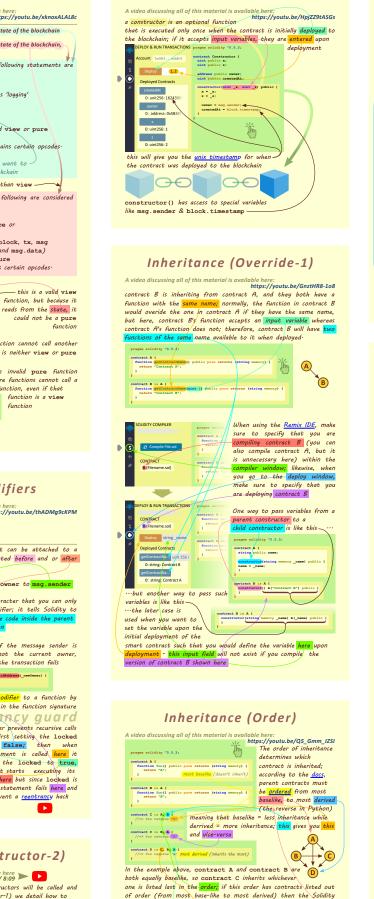
2. Emitting events - also knows as 'logging'
3. Creating other contracts
4. Using selfdestructs
5. Sending Ether via calls
6. Calling any function not marked view or pure
7. Using low-level calls
8. Using inline assembly that contains cartain accords

8. Using inline assembly that contains certain opcodes.

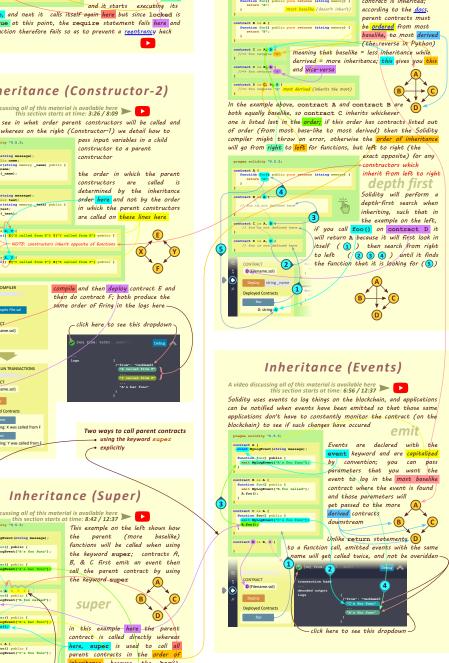
IOTE: this is a function to call when you want to -

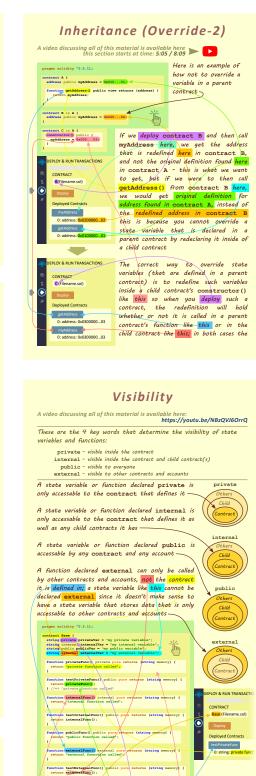
considered modifying the state:

· Writing to state variables



Constructor

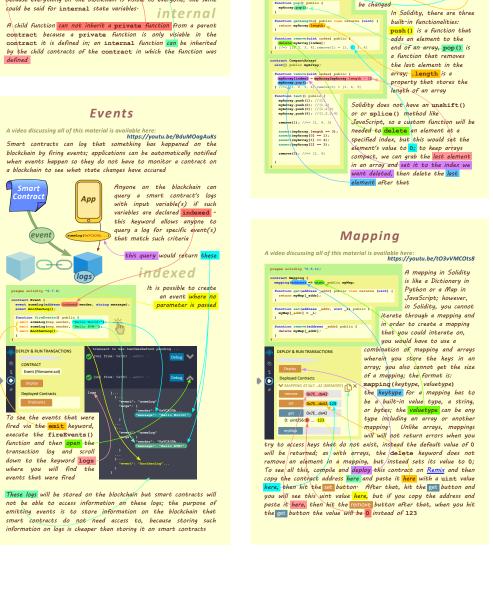


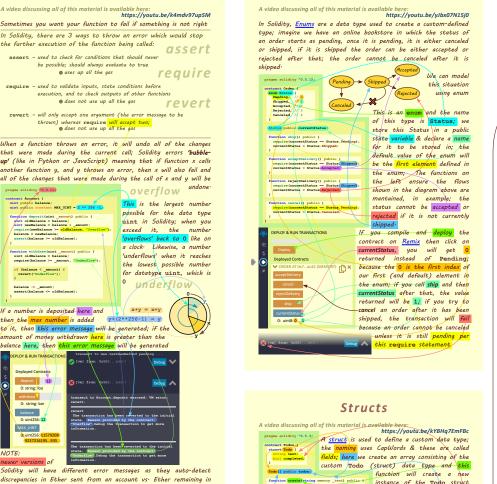


unction like this; then that private function will be accessable

Although private state variables are only accessable inside the

contract, you should not store sensitive information inside them because everything on the blockchain is visible to everyone; the same





Enums

Error

the further execution of the function being called:

assert - used to check for conditions that should never

function withdraw(uint _amount) public {
 uint oldmalance = balance;
 require(balance >= _amount, "Underflow");

if (balance < _amount) (
 revert("Underflow");</pre>

balance -- _amount; assert(balance <- oldBalance);

that same account

Looping

Loops in Solidity are structured just like they are in JavaScript

ion pay() public (
(uint i = 0; (i < *harebolders.length) i++) (
and Ether to such sharebolder

100 iterations | here, this i.

Arrays

arrays are declared with type, accessability, and name; they can be either dynamically sized like this or fixed sized like this; the actual

, size = 10) once the size of the array has been declared, i

and the number of elements in it ca

open like this where you can have unlimited iterations

opposed to leaving them

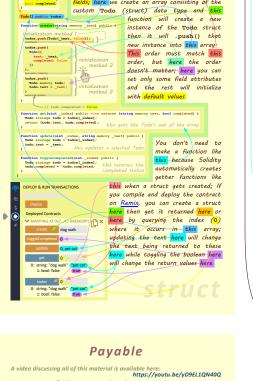
uses up all the gas

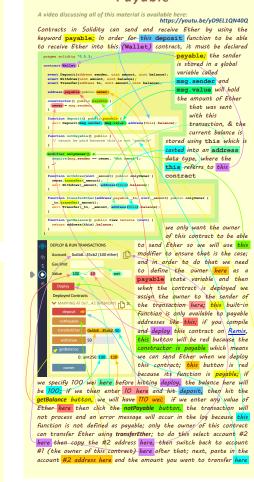
require — used to validate inputs, state conditions before execution, and to check outputs of other functions

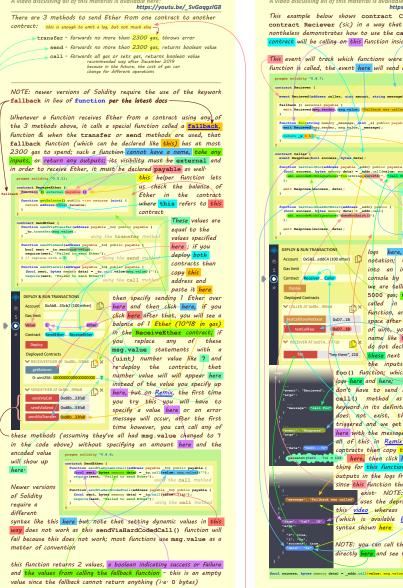
revert - will only accept one argument (the error message to be

thrown) whereas require will accept two;

does not use up all the gas







Sending Ether

nction getBalance() public view return address(this).balance;

unction sendVisSend(address payable bool sent = _to.send(msg.value); require(sent, "Failed to send Ether

matter of convention

enit Log(gasleft());

unction getBalance() public v return address(this).balance

Fallback Function

Fallback functions like this have no names, inputs, or outputs and

must be declared external; you can write regular code inside the fallback function like this but it is reccomended that you limit the amount of code in fallback functions because the function can fail if

uses too much gas which means you won't be able to send Ether a contract using the send or transfer methods; fallback

7. When you call a function

that does not exist in

2. When you send Ether to

the send, transfer, or call methods

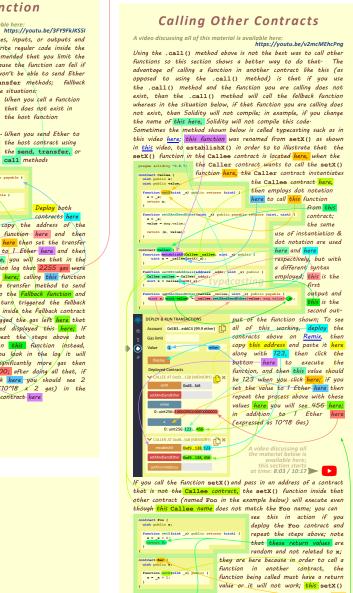
1 Ether here and then

click here, you will see that in th

cauch mare, you will see that in the transaction log that 2255 gas were recorded there; calling this function used the transfer method to send there to the Fallback function and this in turn triggered the fallback districts in incidents callback extracts.

nich logged the gas left here that

300; after doing all that, if



function located inside this

Bar contract for example,

cannot be called by the

Caller function above (if

you repeat the steps in

section) because there is no return value located inside the Bar contract's setX() function

