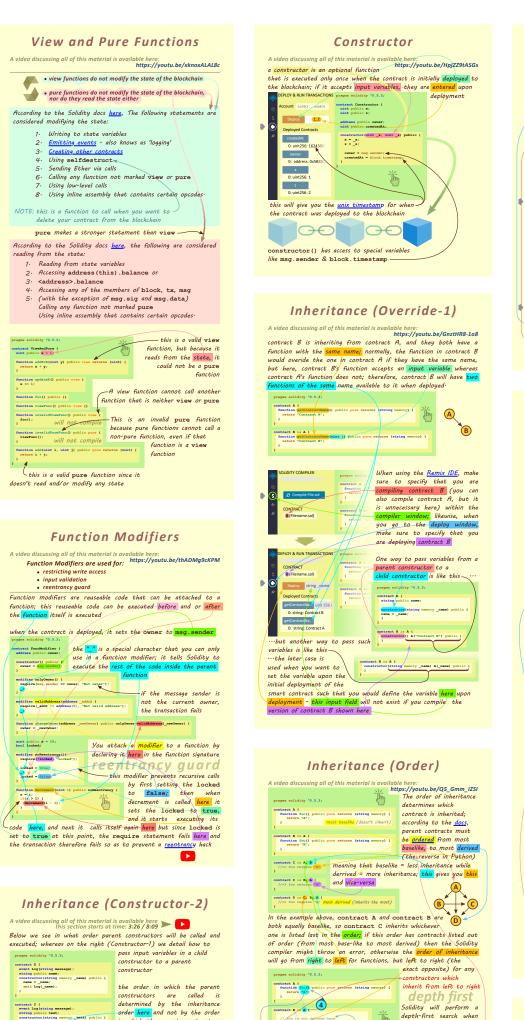


STEP 1: Write a smart contract in this file explorer————

contract SimpleStorage {
 string public text;

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

variables; we do this by using the



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

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 -

1. Reading from state variables
2. Accessing address (this).balance or

Calling any function not marked pure

this is a valid pure function since it

ne function itself is executed

Function Modifiers

function change/yesr(eddress newOwner) public onlyOwner validaddress (newOwner) ( owder = newOyner;

video discussing all of this material is available here this section starts at time: 3:26 / 8:09

text

O: string: Y was ca

constructor to a parent

are called on these lines here

same order of firing in the logs here-

Inheritance (Super)

-click here to see this dropdo

here, X, Y, & Z are hypothetical

pure makes a stronger statement than view —

<address>.balance
Accessing any of the members of block, tx, msg

Using inline assembly that contains certain opcodes.

5. (with the exception of msq.siq and msq.data)

considered modifying the state:

reading from the state:

allowed and "[ ]" is not reccomended.

more reliable (as an input variable) and avoid the

we will also get compile errors

problem with gas (discussed above) is to put an upper-bound to the array size that will in turn limit the amount

We can call firstFunc and

then call secondFunc seperately, or we can just call thirdFunc instead

Here we are assigning the outputs of the

constructors of multiple parent
contracts ··· this is another way

to call the parent constructor inside the constructor of the child contract; here we do not put commas between contracts when listing them

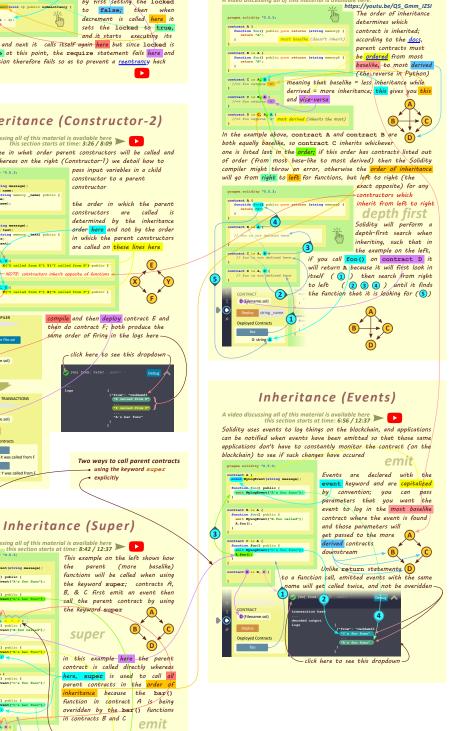
here the constructor is accepting two

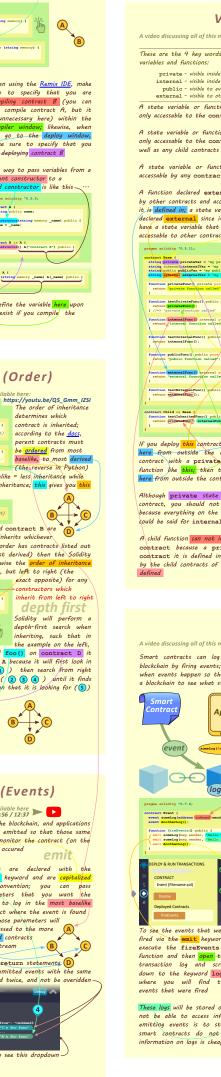
compile and then deploy contract D, and

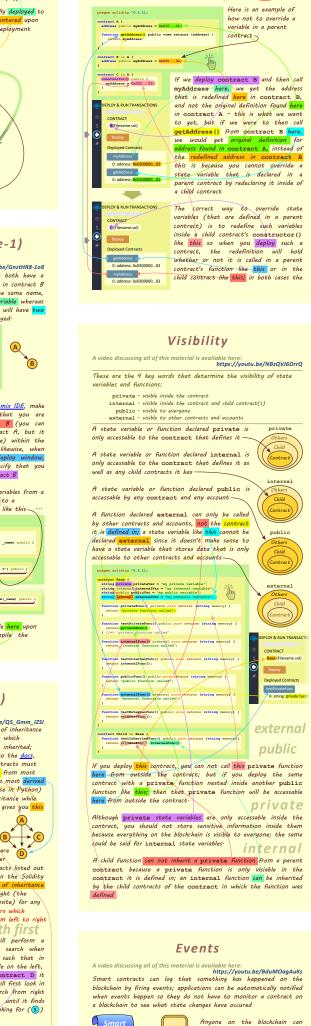
then upon deployment, set the string\_name to "foo" and the string\_text to "bar"

Calling the state variable name returns the string "£00" and calling the state variable text returns the string "bax"

· Writing to state variables







Inheritance (Override-2)

Here is an example of how not to override a

ent contract by redeclaring it inside of

vailable here: https://youtu.be/NBzQVJ6OrrQ

video discussing all of this material is available here this section starts at time: 5:05 / 8:09

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

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

discrepancies in Ether sent from an account vs· Ether remaining in

Looping

Loops in Solidity are structured just like they are in JavaScript

than you are expecting them to because that would use more gas than you were expecting to be used; this you were expecting to be used; this

ion pay() public (
(uint i = 0; (i < third black line) i++) (
and Ether to such sharsholder

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

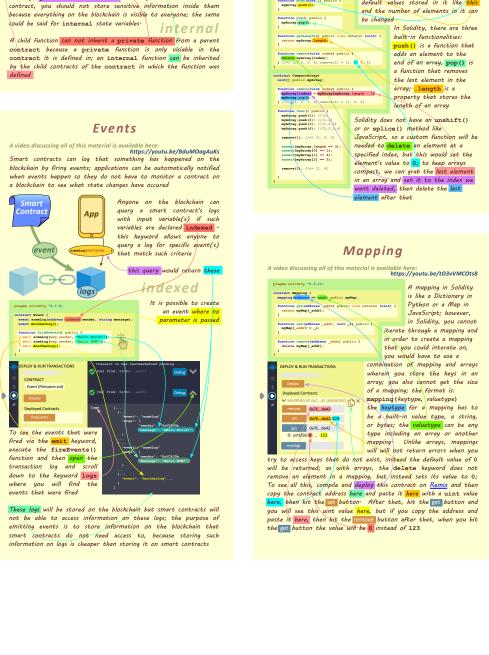
, size = 10) once the size of the array has been declared, i

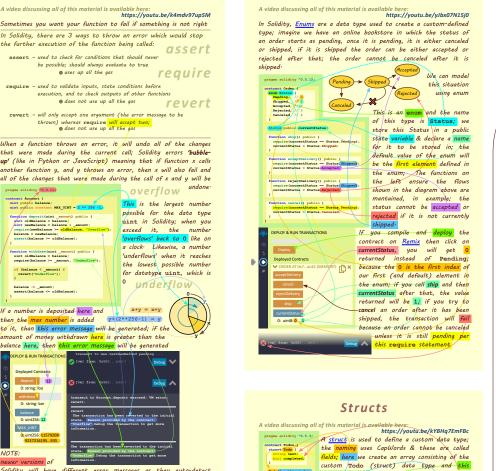
open like this where you can have unlimited iterations

opposed to leaving them

thrown) whereas require will accept two;

does not use up all the gas





Enums

Sending Ether

There are 3 methods to send Ether from one contract to another

transfer - forwards no more than 2300 gas, throws error

call - forwards all gas or sets gas, returns boolean value recommended way after December 2019 because in the future, the cost of gas can change for different operations

NOTE: newer versions of Solidity require the use of the keywork

where this refers to this

These values are

Whenever a function receives Ether from a contract using any

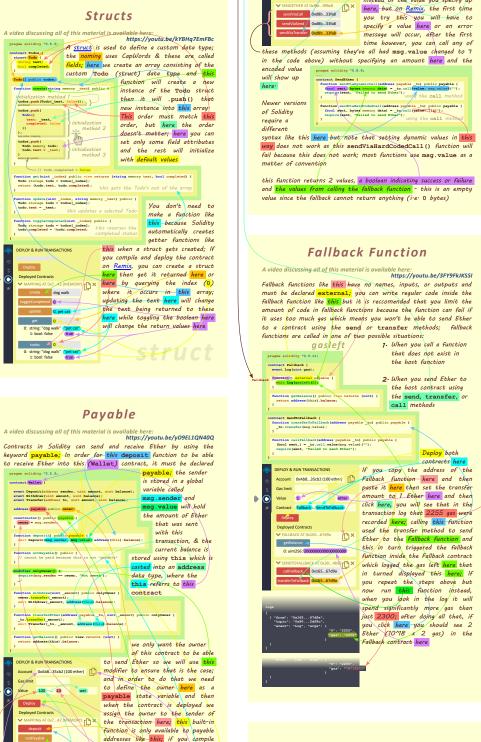
-fallback in lieu of function per the latest docs

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

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

unction sendViaCall(address payable \_to) public payable (
(bool sent, bytes memory data) = \_to.cali.value(msg.value)("")
require(sent, "Failed to send Ether");

unt OxAbs. 35cb/(100 ether) here and then click here, if you will see a

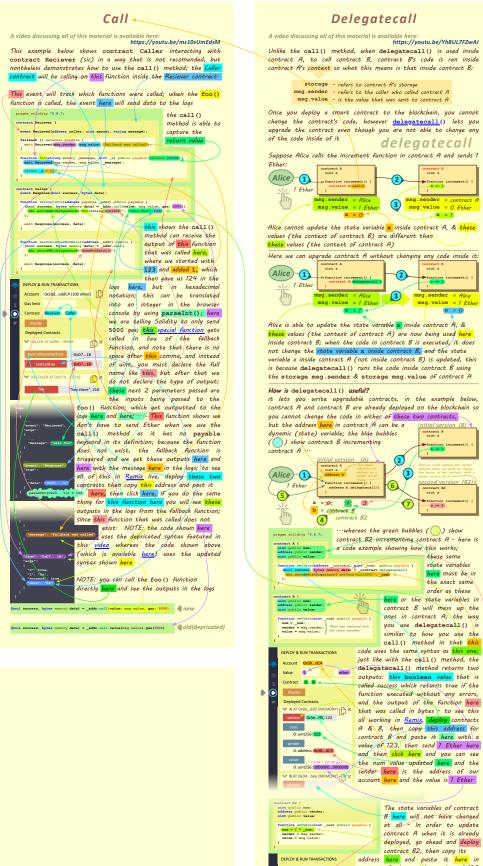


and deploy this contract on Remix

not process and an error message will occur in the log because this function is not defined as payable; only the owner of this contract can transfer Ether using transferther; to do this select account #2 here then copy the #2 address here, then switch back to account #1 (the owner of this contract) here after that; next, paste in the

account #2 address here and the amount you want to transfer he

owner







github.com/Richard-Burd/solidity-sandbox last updated @9:22am on 20/August/2021 by Richard Burd rick.a.burd@gmail.com