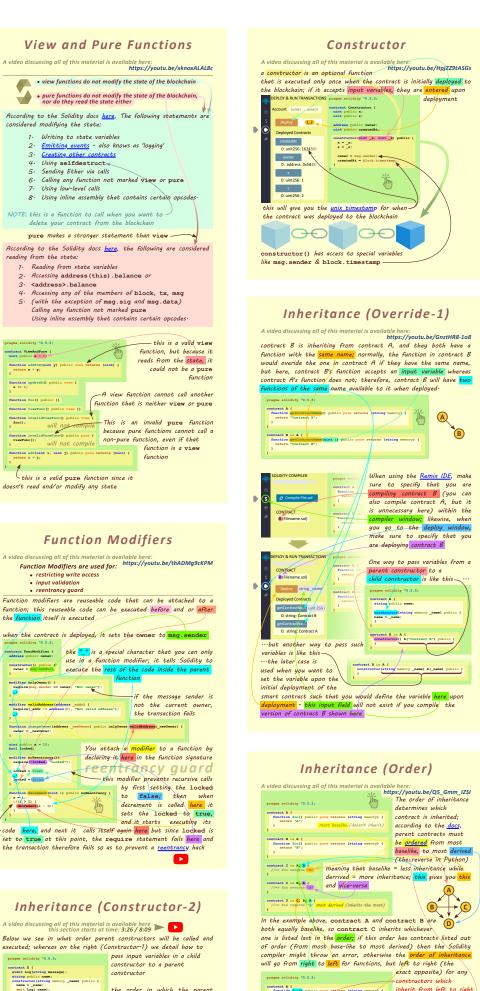


STEP 1: Write a smart contract in this file explorer————

contract SimpleStorage {
 string public text;

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

variables; we do this by using the



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"

text

O: string: Y was ca

here, X, Y, & Z are hypothetical

· Writing to state variables

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

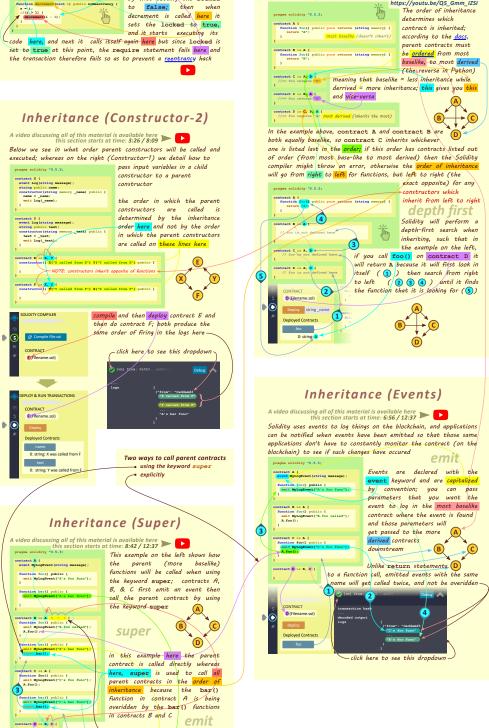
this is a valid pure function since it

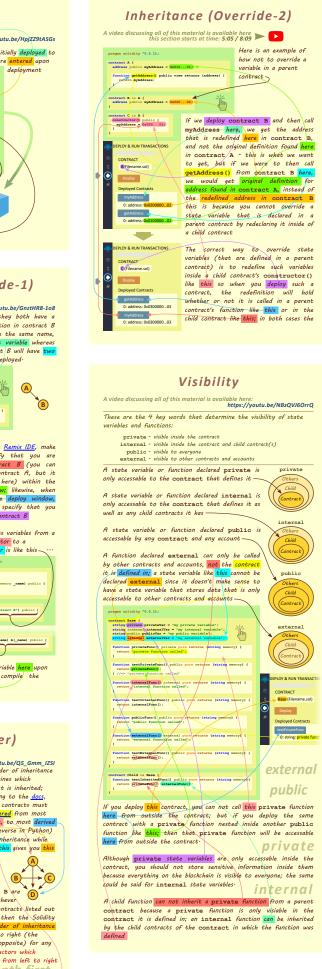
ne function itself is executed

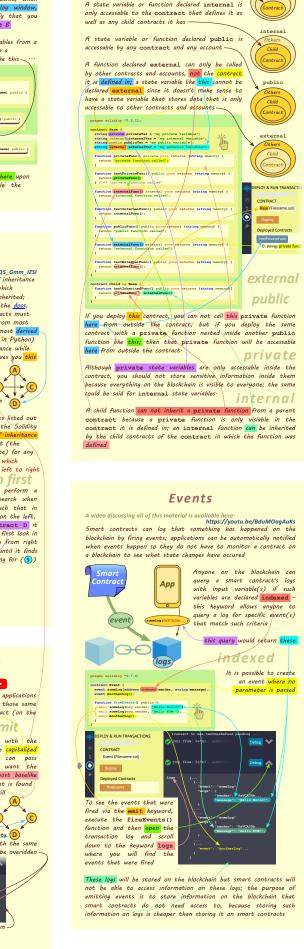
Function Modifiers

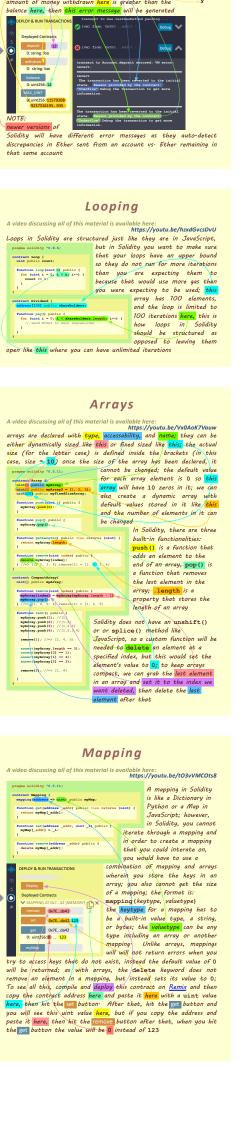
pure makes a stronger statement than view —

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









Error

Sometimes you want your function to fail if something is not right

• does not use up all the gas rever

function ship() public (
require (currentStatus -- Status.Pending);
currentStatus -- Status.Shipped;

tion create string m

unction get(uint \_index) public view Todo storage todo = todos[\_index]; return (todo.text, todo.completed);

function update(uint \_index, string memor Todo storage todo = todos[\_index]; todo.text = \_text;

function toggleCompleted(uint \_index);
Todo storage todo = todos[\_index);
todo completed = !todo.completed;

update 0, pet cat

get 0
0: string: "dog walk" "Pet cat"
1: bool: false true

event Deposit(address sender, uint amount, event Withdraw(uint amount, uint balance); event Transfer(address to, uint amount, uir

unction deposit() public payable ( ment Deposit(msg.sender, msg.value, addre

action withdraw(uint amount) public onlyOwner (
owner transfer(amount);
ent Withdraw(amount, address (this) balance);

notion pathilance() public view returns (unit) (
we only want the owner of this contract to be ab

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 de 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

address payable public owner;

owner

Payable

Contracts in Solidity can send and receive Ether by using th

transaction, & the

data type, where the this referrs to this

and deploy this contract on Remix

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

return myArray length;

assert(myArray.length = assert(myArray[0] == 1) assert(myArray[1] == 4) assert(myArray[2] == 3)

function remove(address \_add delete myMap[\_addr];

0x7E...da42

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

When a function throws an error, it will undo all of the changes

at were made during the current call; Solidity errors **'bubble** 

another function y, and y throws an error, then x will also fail and all of the changes that were made during the call of x and y will be

According ( )

Sylve balance ( )

This is the largest number possible for the data type of the largest possible for the largest

'underflows' when it reaches

he lowest possible numbe

for datatype uint, which is

up' (like in Python or JavaScript) meaning that if function x cal

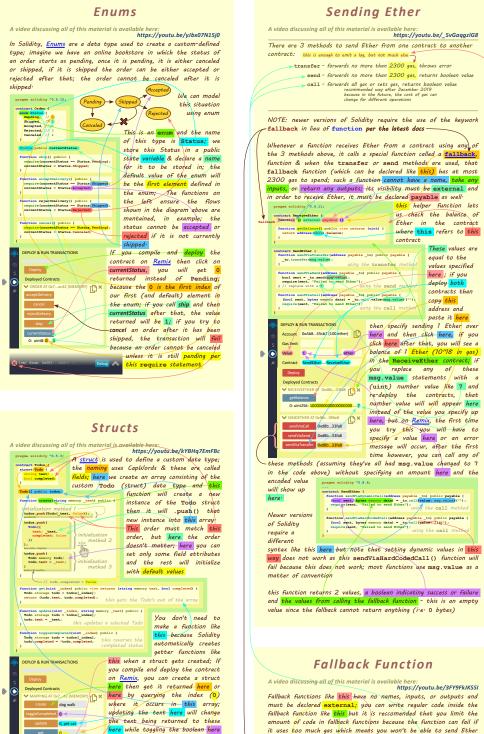
thrown) whereas require will accept two;

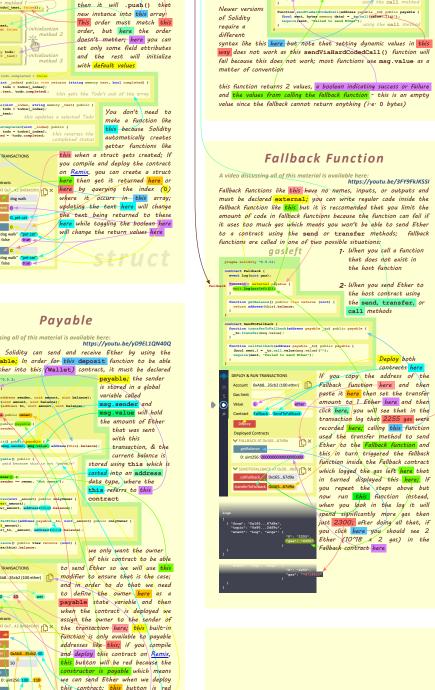
does not use up all the gas

Here is an example of how not to override a

ent contract by redeclaring it inside of

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





where this refers to this

here, but on Remix, the first time you try this you will have to specify a value here or an error message will occur, after the first time however, you can call any of the time time however.

These values are

