Solidity Continued

FinTech
Lesson 20.3

Solidity Continued

By the end of the unit, students will be able to:

- Use `uint` and `int` number types in Solidity and explain when to use each.
- Use global variables to tell the current block number, transaction sender, and transaction value.
- Work with time in Solidity and use time variables to create a timelock.
- Recognize that telling time in Solidity has variability relative to the network's block production time, and static compared to Gregorian calendar time.
- Add conditions to the withdraw function to trigger the timelock from a set threshold.
- Create a contract constructor in Solidity to allow reuse of the contract by preventing hardcoded values.
- Deploy and interact with contracts (Remix + Ganache).

Static Typing Refresher

What is a `uint`?

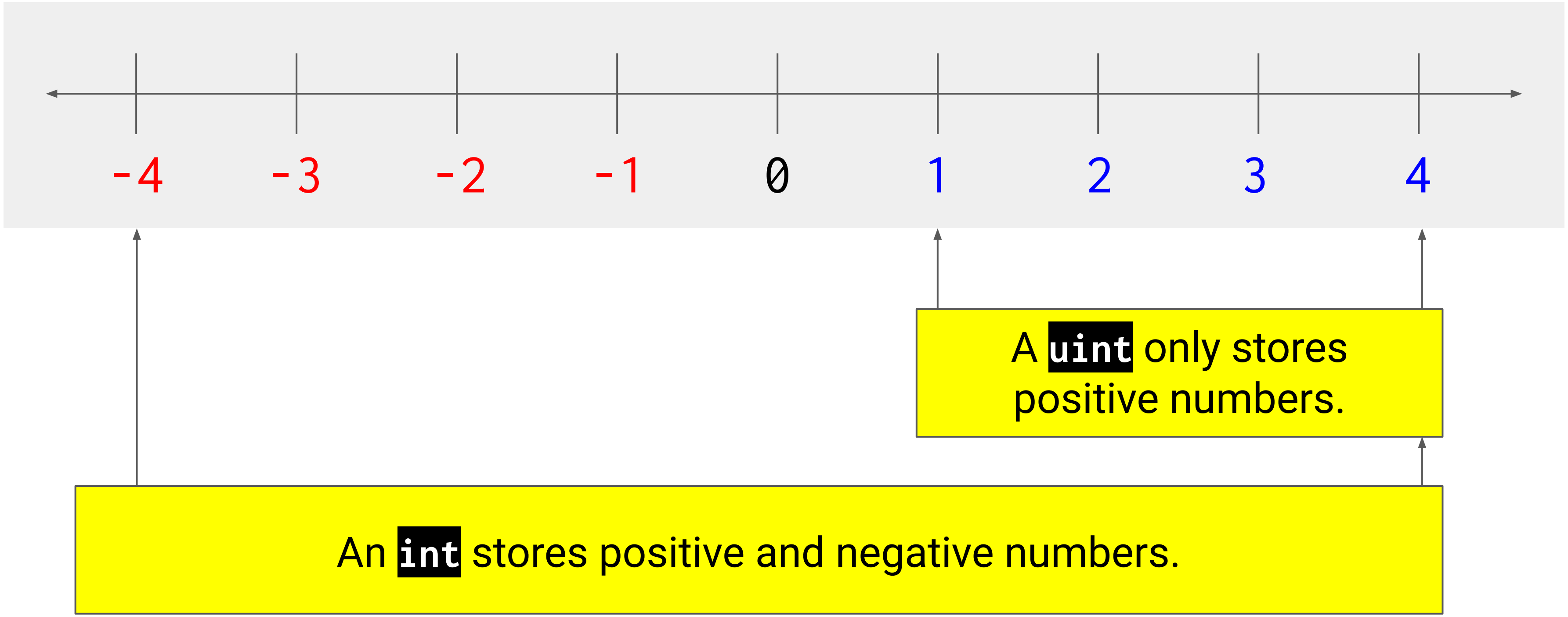
Static Typing Refresher

A `uint` is an unsigned integer that only accepts positive numbers ranging from 0 to 4294967295.

```
uint: 0 to 4294967295
```

What’s the difference between an `int` and a `uint`?

Static Typing Refresher


-4
-3
-2
-1
0
1
2
3
4
A uint only stores positive numbers.
An int stores positive and negative numbers.

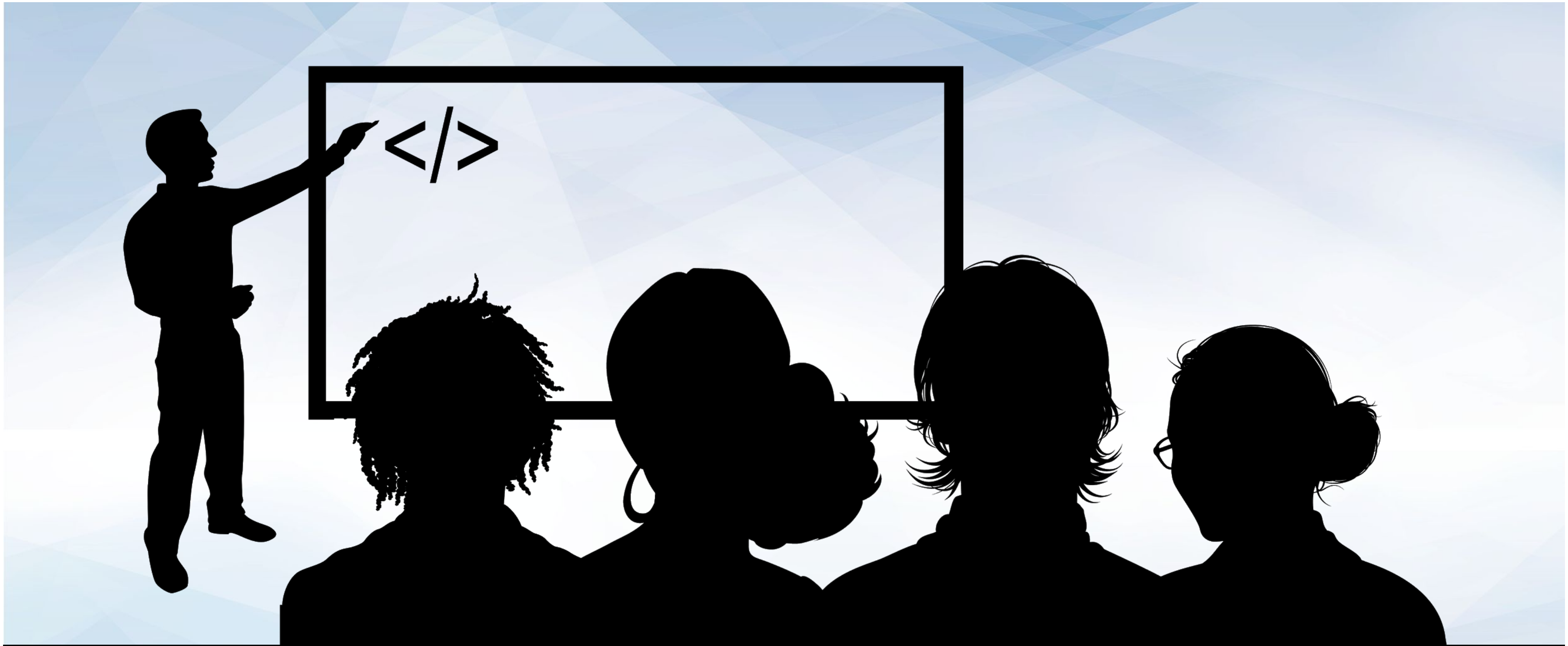
What is a payable address and why is it different from a regular address?

Static Typing Refresher

A payable address is like a normal address type, except it allows the `.transfer` function to be called in order to send it ether.

```
pragma solidity ^0.5.0;
contract JointSavings {
  address payable account_one = 0xc3879B456DAA348a16B6524CBC558d2CC984722c;
  address payable account_two = 0xA29f7E79ECEA4cE30DD78cfeb9605D9aFf5143a5;
  address last_to_withdraw;

 function withdraw(uint amount) public {
   require(msg.sender == account_one || msg.sender == account_two, "You don't own this account!");
   msg.sender.transfer(amount);
 }

 function deposit() public payable {}

 function() external payable {}
}
```

Globally Available Variables and Attributes in Solidity



Instructor Demonstration

Globally Available Variables and Attributes in Solidity

Activity:
Using Global Variables

In this exercise, you'll add the same details using msg and block variables in your contracts.

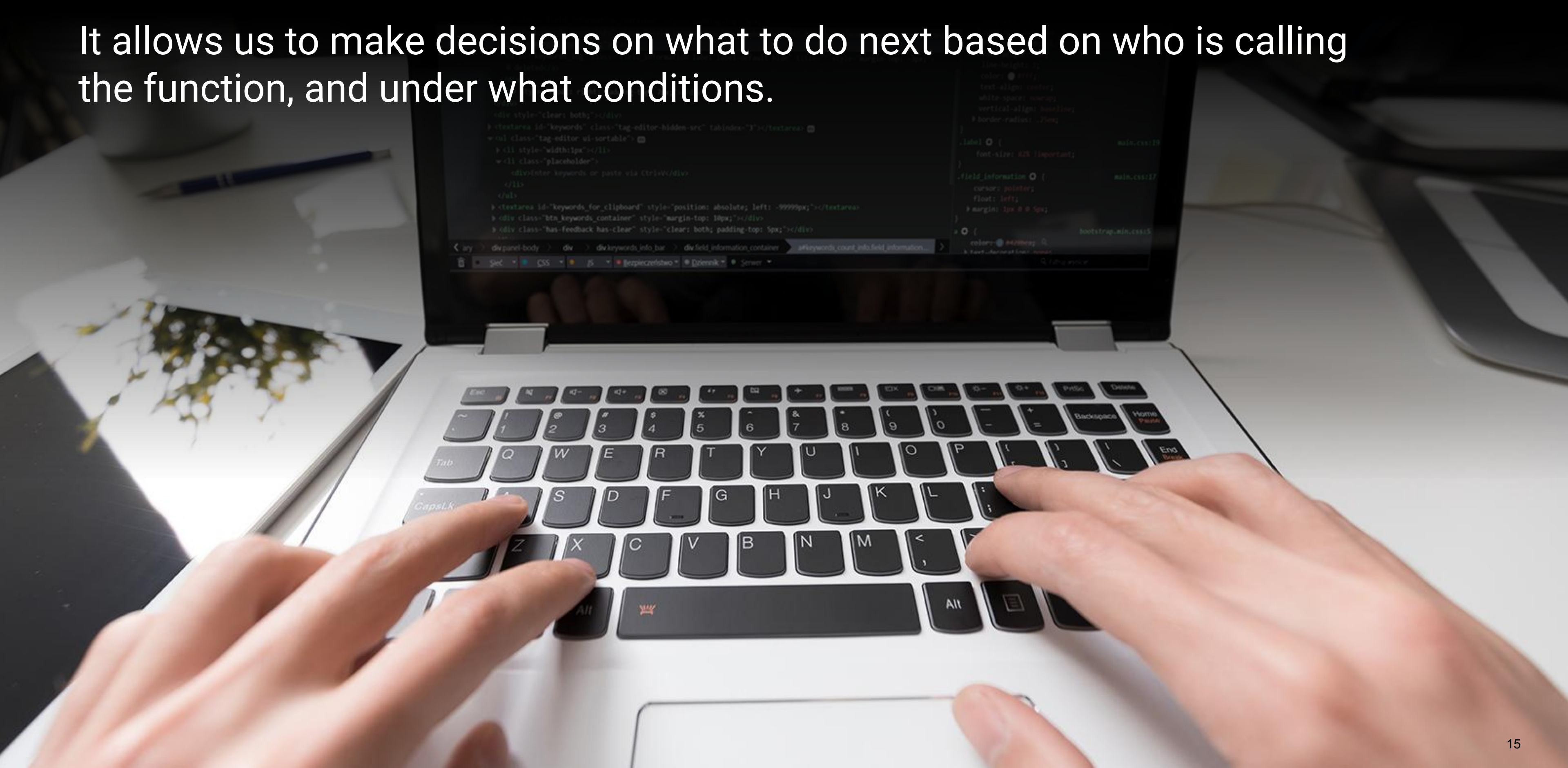
Suggested Time:
10 Minutes

Time's Up! Let's Review.

Why do we access these variables in our contracts?

Globally Available Variables and Attributes in Solidity

It allows us to make decisions on what to do next based on who is calling the function, and under what conditions.

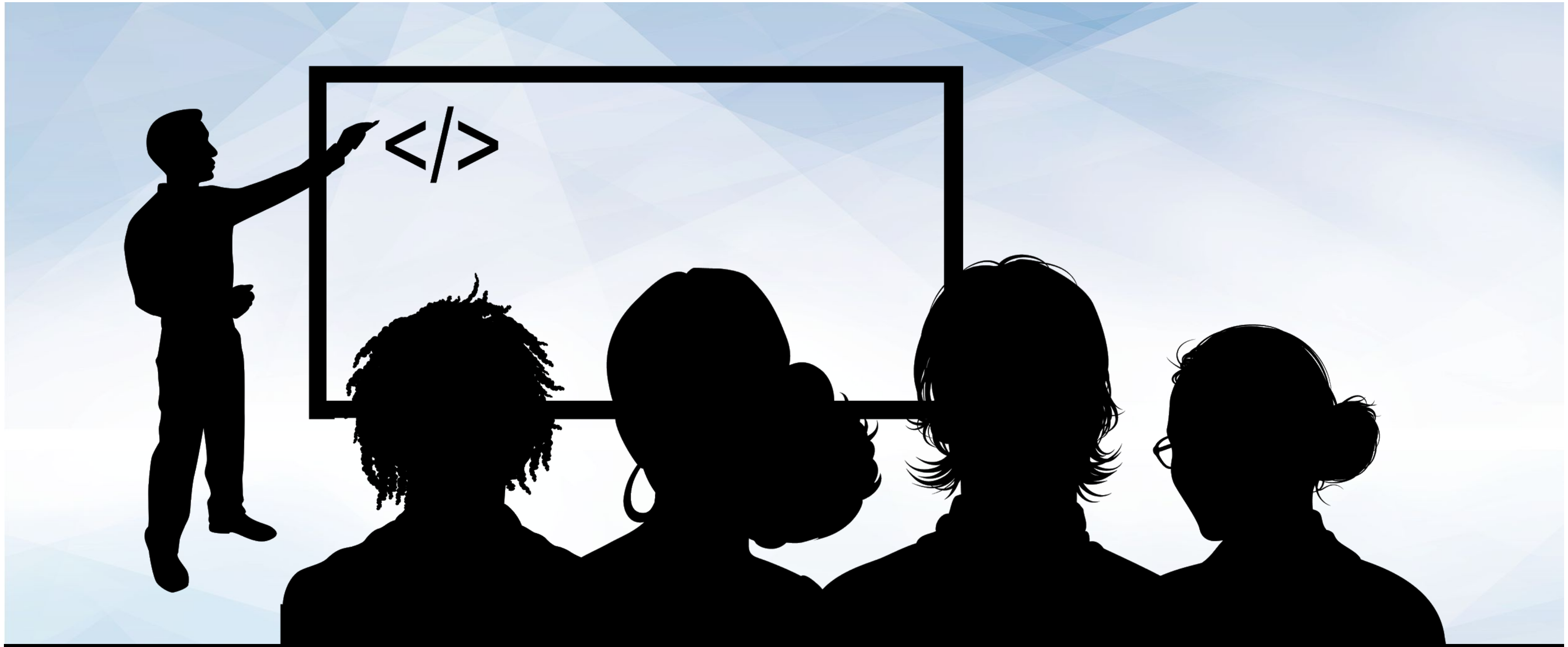


What is `msg.value`?

Globally Available Variables and Attributes in Solidity

Msg. value is the amount of ether that was sent with the transaction, in the smallest denomination, wei.

Telling Time in Solidity



Instructor Demonstration

Telling Time in Solidity

Activity:
Creating a Timelock

In this exercise, you'll add the same timelock to your `JointSavings` contracts.

Suggested Time:
10 Minutes

Time's Up! Let's Review.

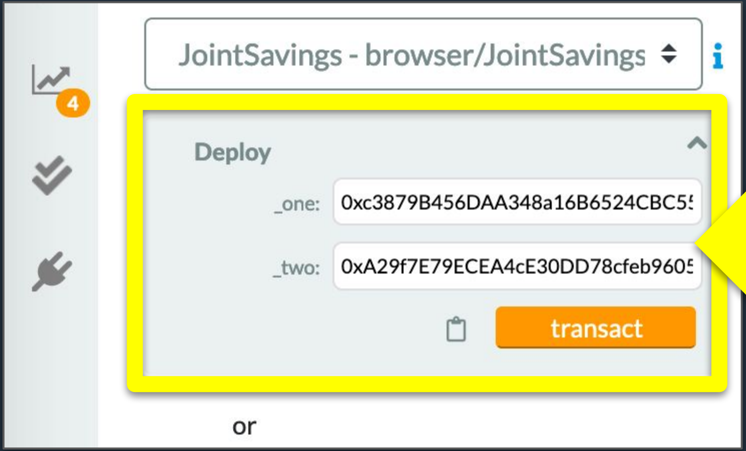
Telling Time in Solidity

For the second half of class, we are going to add a threshold that only locks our savings account when more than 1/3 of the balance is pulled.

```
if (amount > address(this).balance / 3) {
 unlock_time = now + 24 hours;
}
```

Telling Time in Solidity

Then, we will add a special function called a `constructor` that will allow us to deploy this contract with custom values vs. hardcoding things like our account owners directly in the code.

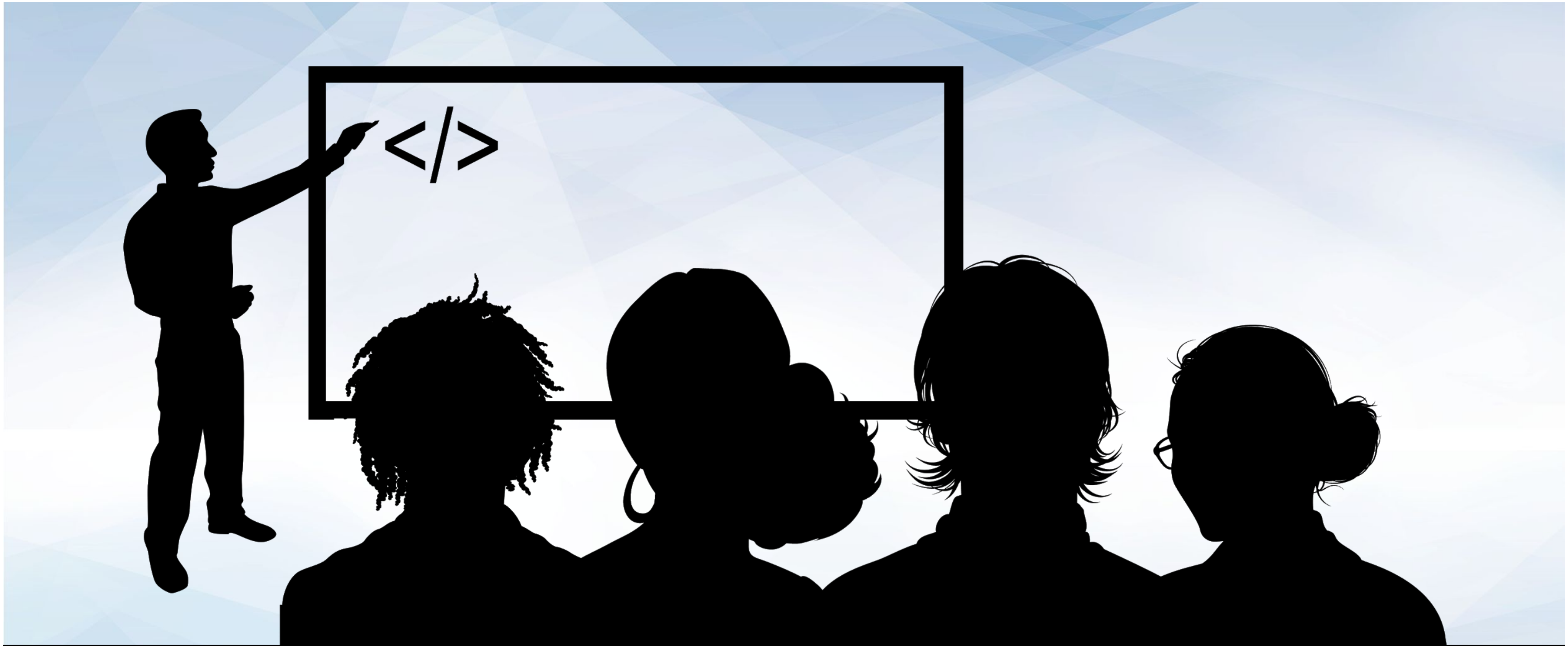


_one: 0xc3879B456DAA348a16B6524CBC5!

_two: 0xA29f7E79ECEA4cE30DD78cfeb960!

```
contract JointSavings {
  address payable account_one;
  address payable account_two;

  address public last_to_withdraw;
  uint public last_withdraw_block;
  uint public last_withdraw_amount;

  address public last_to_deposit;
  uint public last_deposit_block;
  uint public last_deposit_amount;

  uint unlock_time;

  constructor(address payable _one, address payable _two) public {
  account_one = _one;
  account_two = _two;
  }
```

Adding a Withdraw Threshold



Instructor Demonstration

Adding a Withdraw Threshold

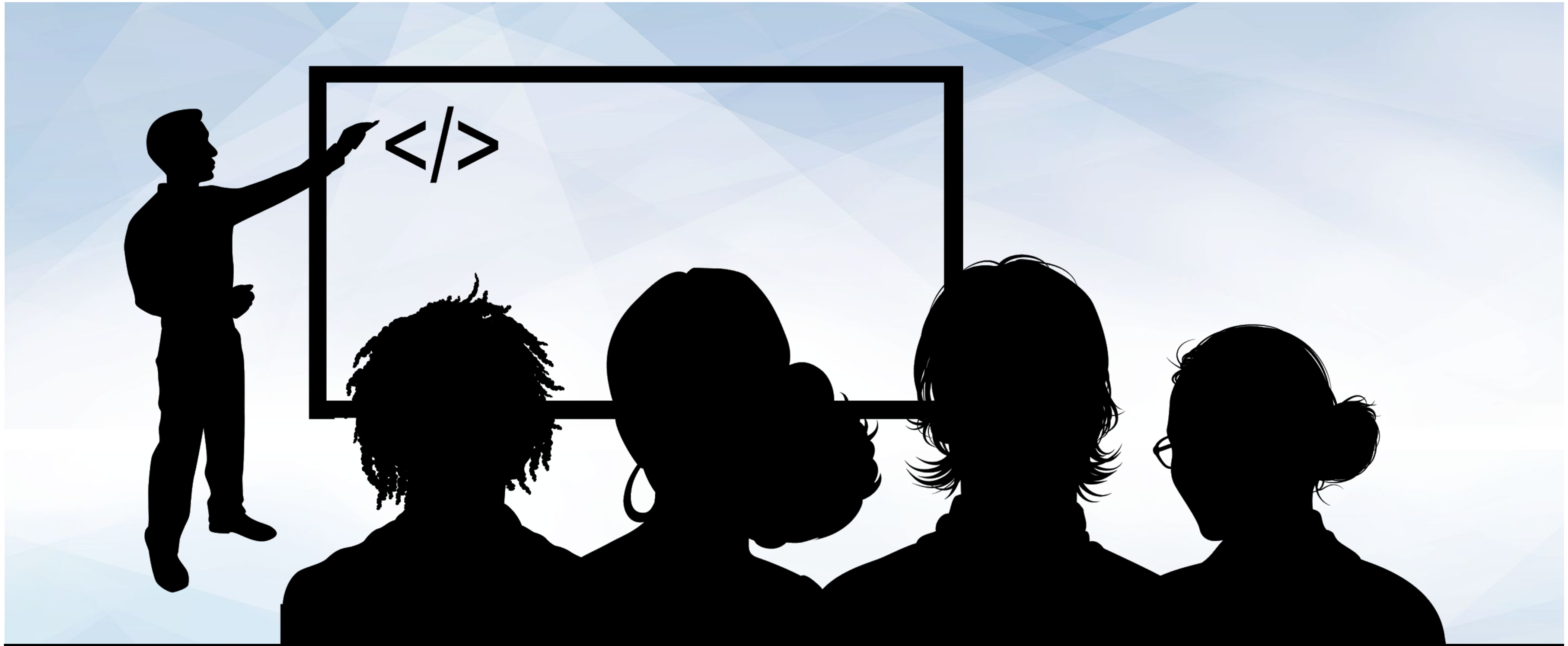
Activity:
Adding the Withdraw Threshold

In this exercise, you'll follow the same steps to add the threshold to your withdraw function’s time lock.

Suggested Time:
10 Minutes

Time’s Up! Let’s Review.

Intro to Constructors



Instructor Demonstration

Intro to Constructors

Activity: Adding a Constructor to the Contract

In this exercise, you'll replace your hardcoded values with a constructor in order to make your contracts reusable and more production-ready.

Suggested Time:
15 Minutes

Time’s Up! Let’s Review.

Deploying and Testing the Contract

Activity: Deploying and Testing the Contract

In this exercise, you'll deploy the current contract, setting the account owners in the constructor in the deployment tab of Remix.

Suggested Time:
15 Minutes

Time’s Up! Let’s Review.

Smart Contracts Review

What are smart contracts useful for?

Smart Contracts Review

We can build any program on top of the blockchain, giving us the ability to write fully decentralized applications.

Why do we use a constructor?

Smart Contracts Review

So we can avoid hardcoded values and reuse our code.

```
contract JointSavings {
 address payable account_one;
 address payable account_two;
 address public last_to_withdraw;
 uint public last_withdraw_block;
 uint public last_withdraw_amount;
 address public last_to_deposit;
 uint public last_deposit_block;
 uint public last_deposit_amount;
 uint unlock_time;

 constructor(address payable _one, address payable _two) public {
   account_one = _one;
   account_two = _two;
 }
// ...
}
```

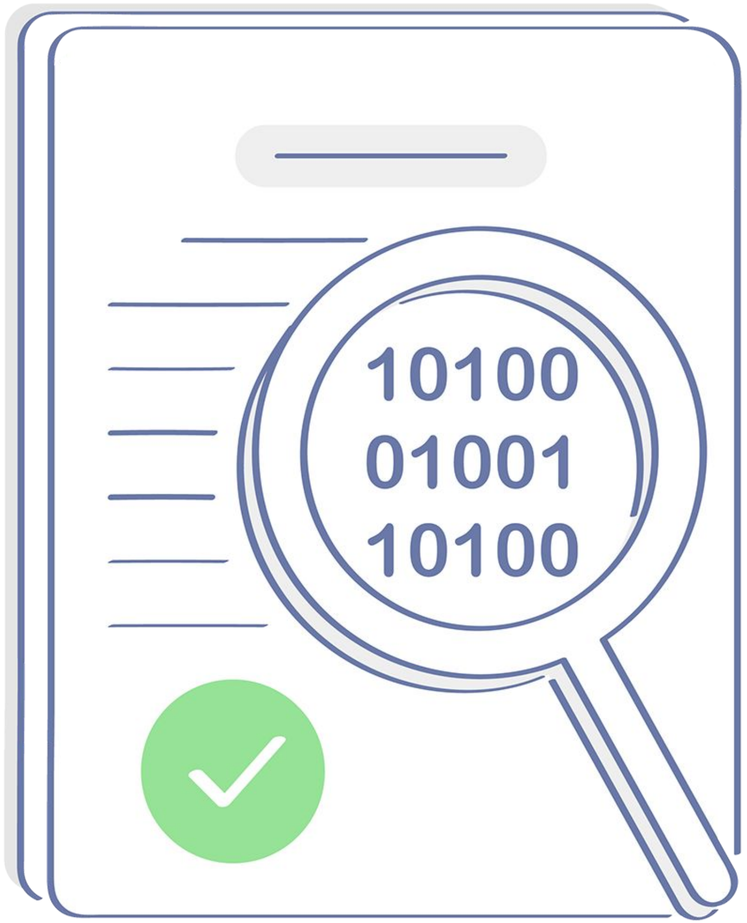
Why do we use strict data types in Solidity?

Smart Contracts Review

We need to be unambiguous in our code, just like we need to be unambiguous in legal contracts.

Smart Contracts Review

It is more efficient, and thus cheaper, to define the types up front, vs. having Solidity figure it out after the fact, spending precious gas.



What nuances are there when it comes to telling time in Solidity/Ethereum?

Smart Contracts Review

When it comes to telling time in Solidity/Ethereum, nuances are:

01

We are limited to static time that is not Gregorian.

02

We are limited to a window of accuracy defined by the average block time.

03

In order to get more accurate time, we need special Oracle contracts.

Questions?

The End