Interacting with Smart Contracts

FinTech

Lesson 20.2

Class Objectives

- Define the `payable` function's role in smart contracts.
- Develop the syntax in Solidity to withdraw and deposit ether.
- Explain the purpose of using the keyword `public` in smart contracts.
- Use the `this` keyword in smart contracts.
- Create a public `getter` function to manipulate trades by sending different denominations of coins.
- Write conditional statements in Solidity.
- Use conditional statements in Solidity in order to obtain the desired results in your smart contract.
- Demonstrate how the `require` function works in Solidity smart contracts.
- Develop a smart contract using the `require` function to enforce smart contract terms.


RECAP

Recap

We defined the basic structure of a smart contract, and wrote a function as well as variables for storing data.

We'll create functions that add functionality to a smart contract.

REMEMBER

In previous units, you wrote Python functions.

Solidity and Business Rules

In Solidity, a developer can add custom business rules to a smart contract.

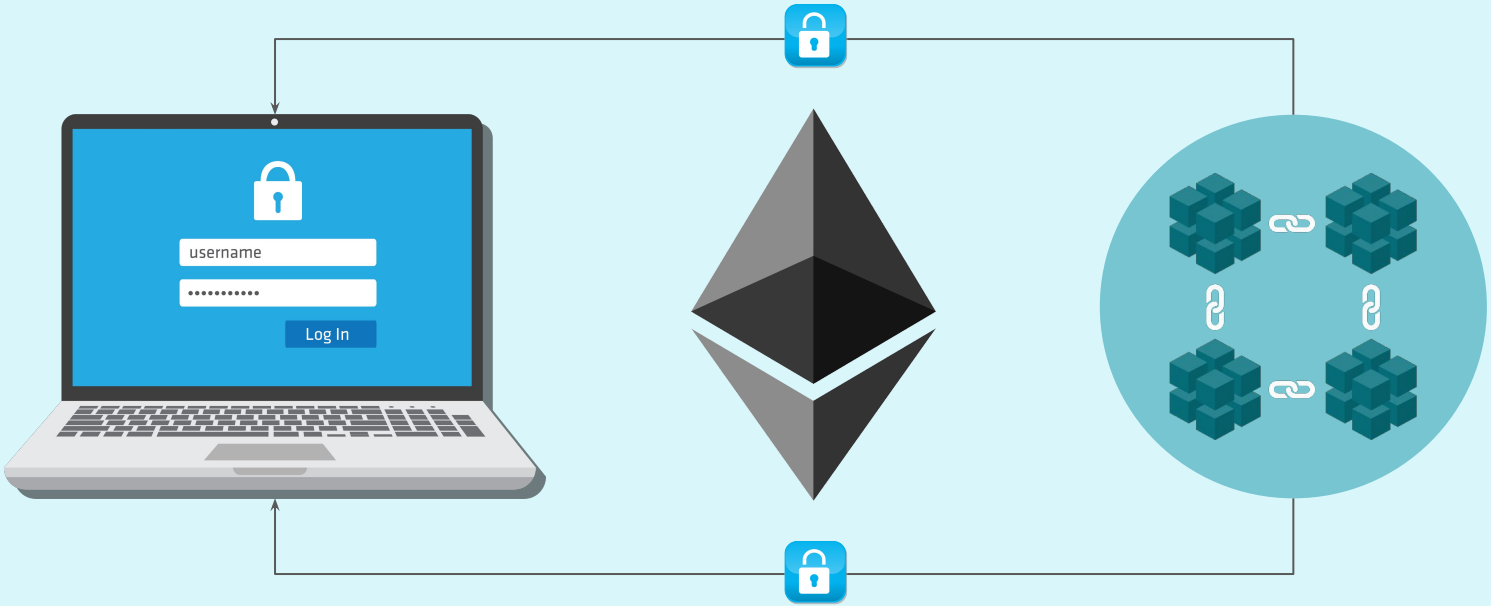
These rules verify whether a person is authorized to make a transaction.

A **buy order** refers to the action of buying a certain amount of financial assets.

The **execution price** is the price at which the trade was executed.

Solidity and Business Rules

In order to enable a smart contract to interact with the blockchain or even with other smart contracts, we need to define a function that captures the business rules that we want to implement.



As we did in Python, we can define functions in Solidity—but with some nuances.

The payable Function

In order to trade, we must be able to send and receive currency.

The currency we're using is ether.

In order for our functions to receive ether, they must be attached to a modifier called `payable`.

`payable` is a reserved keyword in Solidity.

`payable` is a mandatory keyword for trading because, without it, the function will be rejected, and no ether will be sent.

Fallback Function

Usually, there is a `no name` function that accepts ether that is to be sent to a contract.

This is called a **fallback function,** which we'll explain in more detail in the next section.

```
function () payable {}
```

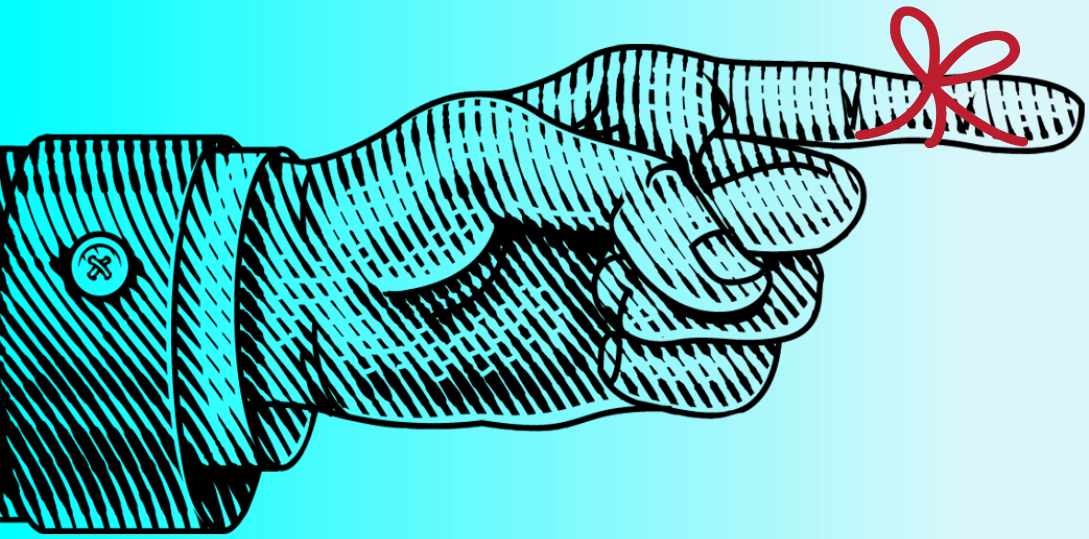
Fallback Function

An exception is when you have more than one `payable` function that is used to perform different tasks, such as registering a deposit to your contract:

```
function deposit() payable {
  deposits[msg.sender] += msg.value;
};
```

Questions?

Making Transactions in the Ethereum Blockchain



Remember:

- Blockchain is a technology that records and shares data.
- By using functions, we can store data in a contract.
- One application of this in the fintech realm is keeping track of asset transactions.

Making Transactions in the Ethereum Blockchain

We can add special functions to smart contracts that will allow us to deposit or withdraw ether.

Ether is the native cryptocurrency of the Ethereum network and one of the largest cryptocurrencies by market capitalization.

Besides depositing and withdrawing ether, we'll also build functions that allow us to catch ether that's sent from outside a function call.

These functions make up the core building blocks for managing digital assets in smart contracts.

Sending currency is an important part of a blockchain developer's role.

It allows them to be an asset when dealing with transactions and building applications on the Ethereum blockchain.

Questions?

Withdraw Ether from a Smart Contract

The Withdraw Function

The `withdraw` function allows us to take ether from a smart contract.
It accepts the following arguments:

`uint`	An argument that represents the amount of ether, in its smaller denomination named wei, that we want to withdraw.	We name it `amount`
`address`	An argument that represents the Ethereum address that we want to send the funds being withdrawn to.	We name it `recipient`

Solidity's Address Types

Pay special attention to the new modifier, named `payable` that we use for the `address` argument.

By setting an address or function as `payable` we unlock special functions that allow us to collect and manage ether.

`address`	Can store a 20-byte value, which is the size of an Ethereum address.
`address payable`	The same as `address` but includes functions, like `send` and `transfer`, that allow us to perform operations with ether.

Class Slack Channel:

You are encouraged to review the Solidity documentation.

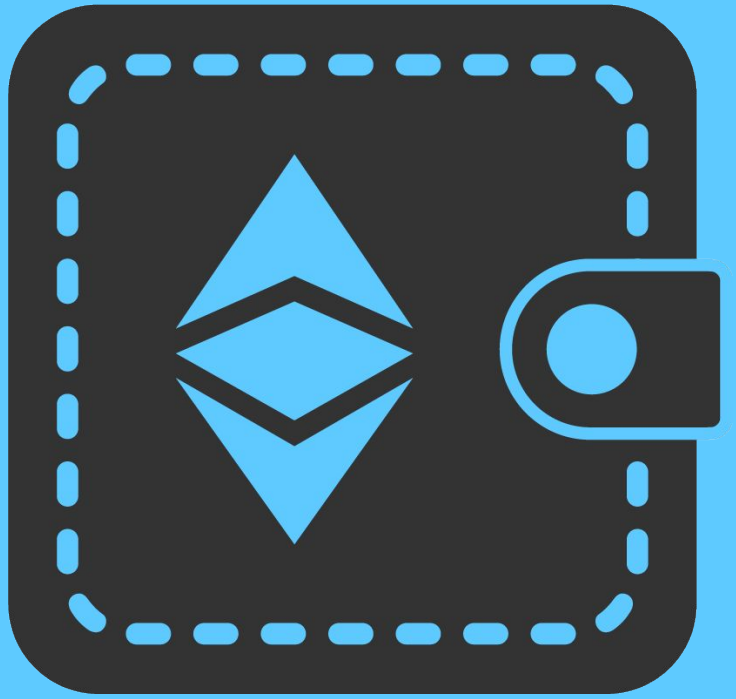
Questions?

Deposit Ether to a Smart Contract

Deposit Ether to a Smart Contract

Each time you deploy a smart contract, a public Ethereum address is assigned to it.

- A smart contract can store and send ether like a cryptocurrency wallet with its Ethereum account address.
- It’s up to developers to create functions that manage this address, just like we did with the withdraw function.



Collect Ether Without a Deposit Function

Collect Ether Without a Deposit Function

We can make sure that ether gets sent to the contract without using the deposit function (that is, by sending ether directly to the contract's address).

The contract still collects the ether in the contract wallet.

It's important not to lose even one wei.

Fallback Function

The function without a name, known as a **fallback function**, is used in two scenarios:

01

If the function identifier doesn't match any other function in the contract.

02

If the sending function doesn't supply any data, so we have to add the `external` keyword so that other contracts or transactions can call this contract.

We also add the `payable` keyword so that the contract can collect any amount of ether that gets sent to it via the contract address.

Fallback Function

If we don't add the fallback function,
and ether gets sent to our contract address, the ether will be returned.

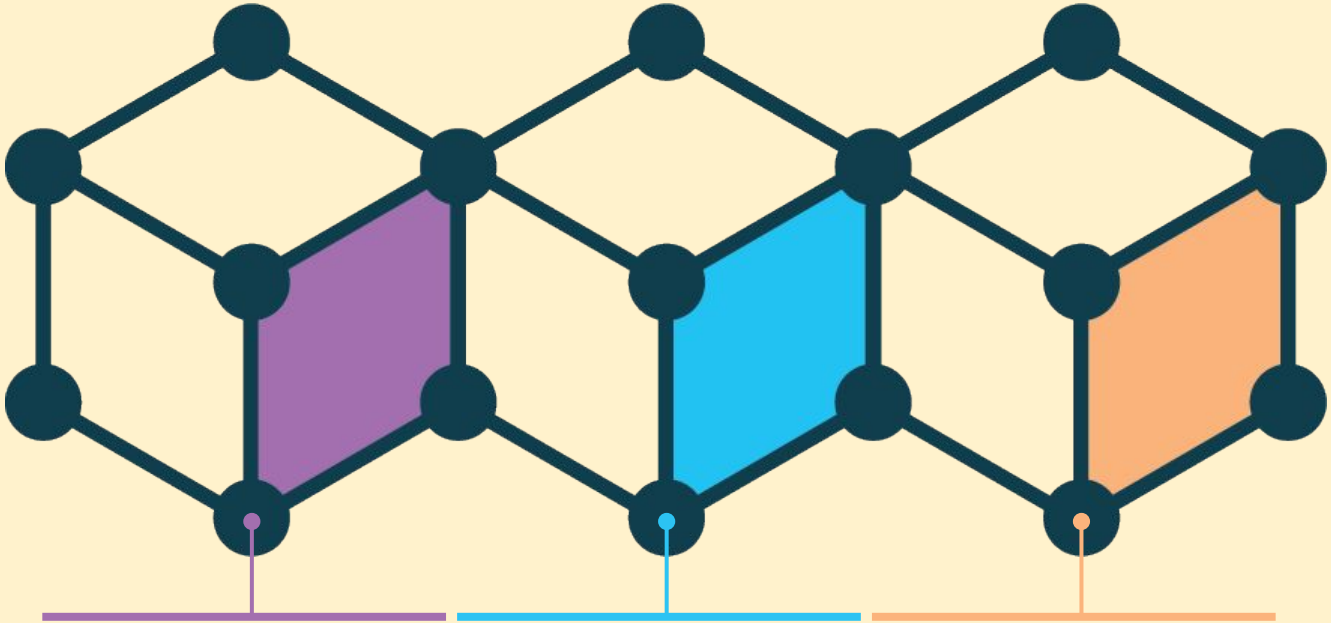
This forces other users to send ether via the `deposit` function.

Paying Gas to Run Your Contracts

Moving ether in the Ethereum blockchain implies a cost that’s known as gas.

Gas is a kind of fee that a user pays for each instruction that the Ethereum Virtual Machine runs.

Paying Gas to Run Your Contracts



Ethereum blockchain users pay fees that are based on running different computations.

Each computation has a different gas cost.

Each time that you run a function within a smart contract, you have to pay a gas fee to run that function in the blockchain.

Paying Gas to Run Your Contracts

The blockchain miners determine the precise gas fee.

They occasionally update the gas fee when certain code executions become too expensive for average nodes to process.

This is to keep the blockchain network fair.

Paying Gas to Run Your Contracts

Gas is essential to the Ethereum network.

It's the energy that it needs to work—
in the same way that a bulb needs electricity to light up.

Class Slack Channel:

Ethereum's documentation on gas and fees
is a good resource to learn more about how gas costs operate.

What if we don’t have enough gas to complete a transaction?

Do we lose the gas that we were charged?

We do lose the gas that we were charged. But, the transaction will be reversed, and we'll get our ether back.

Questions?

Time to Code

Implement Ether Management Functions

Suggested Time:

15 minutes

A **remittances application** allows people to both deposit ether into a smart contract and withdraw funds to send to friends and family members that have an Ethereum address.

Questions?

Time to Code

Trading and Savings Contracts

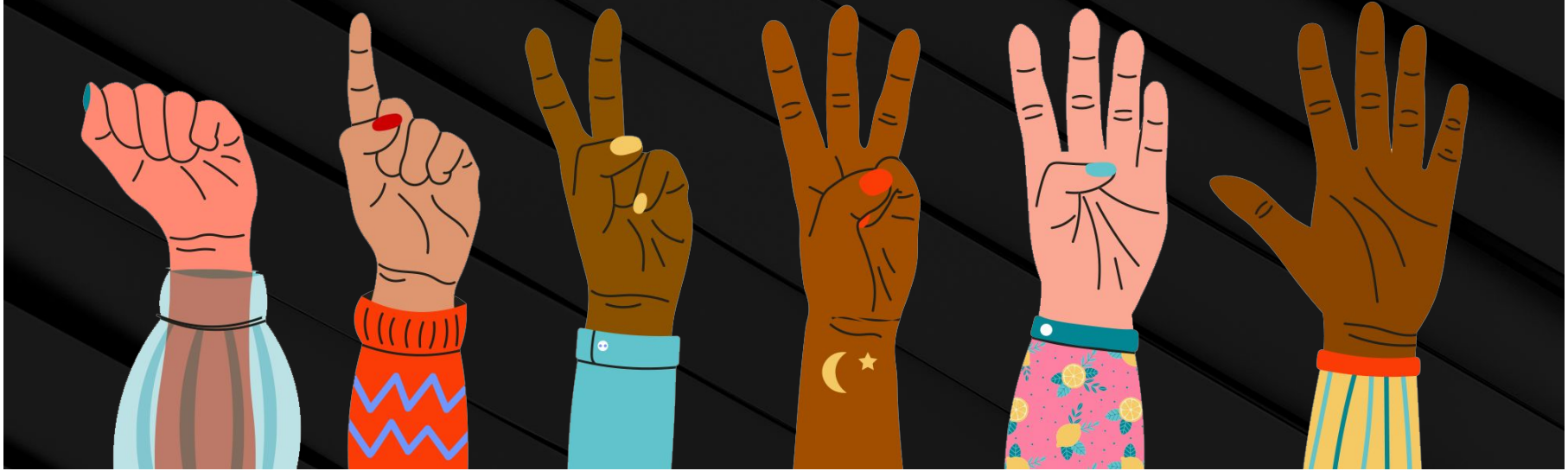
Suggested Time:

30 minutes

Time's Up! Let's Review.

FIST TO FIVE:

Review Trading and Savings Contracts



Questions?


Break

Instructor Demonstration

Decision-Making in Smart Contracts

Decision-Making in Smart Contracts

Solidity provides a set of statements to implement conditionals.
These statements can be used to define the terms of a smart contract.

conditionals	Can be used to define the terms of a smart contract.
`if` statement	Can be used to implement decision-making in smart contracts.
`else` statement	Can be used when the condition of the `if` statement is false.
stacked conditionals	Can be used to “stack,” or chain, conditionals together.

Decision-Making in Smart Contracts

The relational operators that the conditions use match the ones in Python.

<=	Less than or equal to
<	Less than
==	Equal to

!=	Not equal to
>=	Greater than or equal to
>	Greater than

Only the `not` operator differs. In Solidity, we use the exclamation point (`!`) to denote the negation of a conditional statement.

Questions?

Time to Code

Trade Controllers

Suggested Time:

20 minutes

Time's Up! Let's Review.

Questions?

Instructor Demonstration

The require Function

Solidity provides an alternative to using conditional statements to define the contract terms before allowing its fulfillment: it's called the `require` function.

The require Function

As a fintech professional, you can strengthen your contract policies by using the require function. Use cases include:

01

Verifying that a recipient is someone you know after completing a withdrawal transaction.

02

Verifying that the sender smart contract has enough ether to cover the requested amount.

Class Slack Channel:

You are encouraged to review the Solidity documentation on error handling.

Time to Code

Enforcing Contract Terms

Suggested Time:

25 minutes

Time's Up! Let's Review.

Questions?


The End