Smart Contracts Fundamentals

FinTech

Lesson 20.1

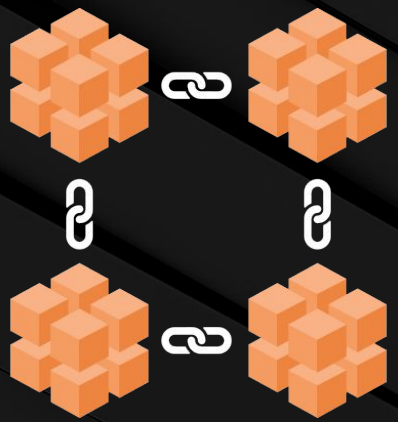
© 2022 edX Boot Camps LLC. Confidential and Proprietary. All Rights Reserved.

Class Objectives

- Explain what a smart contract is and explain its applications in fintech and other business areas.
- Explain how Solidity works and how it differs from Python as a programming language.
- Explain how the Ethereum Virtual Machine (EVM) is an isolated environment and how Solidity code can access only on-chain data.
- Recognize that a smart contract is a program that runs in the EVM.
- Identify the kinds of smart contracts and their top use cases.
- Explain how Remix supports blockchain development.
- Create three types of functions in Solidity: a deposit function that adds ether, a withdraw function that withdraws ether, and a fallback function that captures ether.
- Create getters and setters, including their return types, in Solidity.

WELCOME

Introduction to Smart Contracts

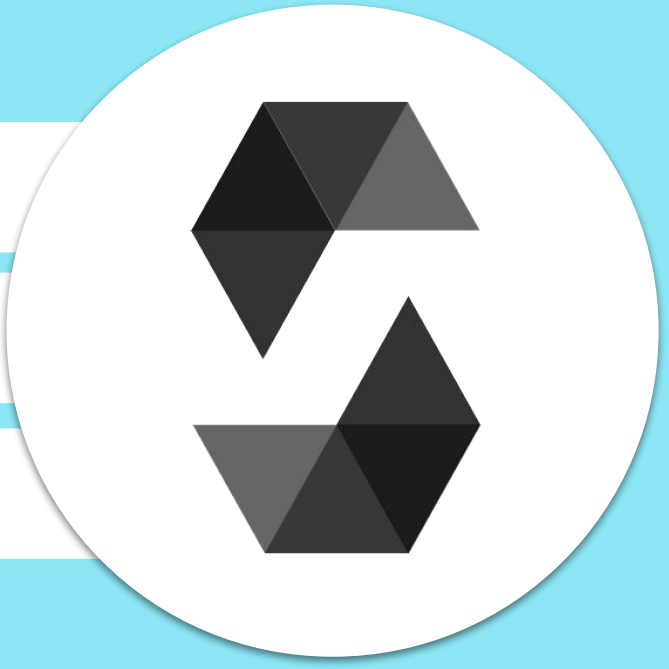


Remember, blockchain is used to establish trust with a network that maintains an immutable ledger.

Introduction to Solidity

We are going to use **Solidity**, the most popular programming language to create and deploy smart contracts in Ethereum.

- Solidity enhances our transaction processes.
- Solidity allows us to establish the terms of a financial contract.
- Solidity gives us the ability to create applications and services on the blockchain.



Introduction to Smart Contracts

We can use smart contracts to define the parameters for a purchase, an auction, voting, banking transactions, and more.

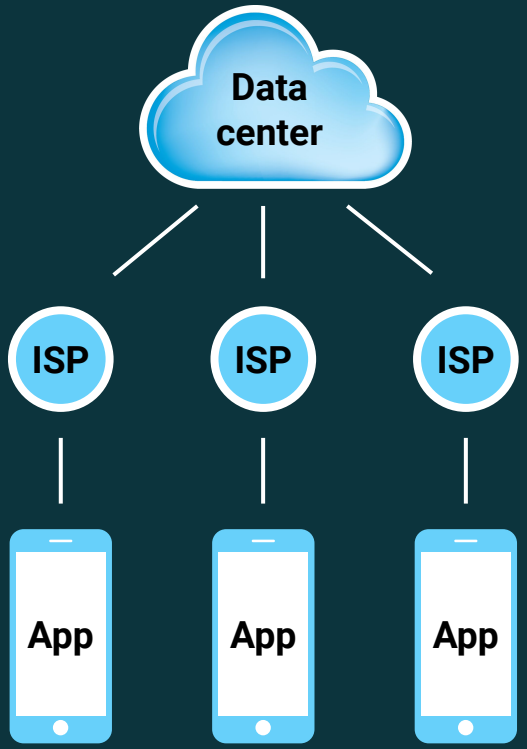
So, a developer that can program in Solidity is valuable to institutions that are involved with blockchain transactions.

Smart contracts are like computer programs that can run on a blockchain. This means that people can use them to build decentralized applications (dApps) that can run code in a trustworthy way.

dApps are software that has decentralized operations and that uses decentralized storage.

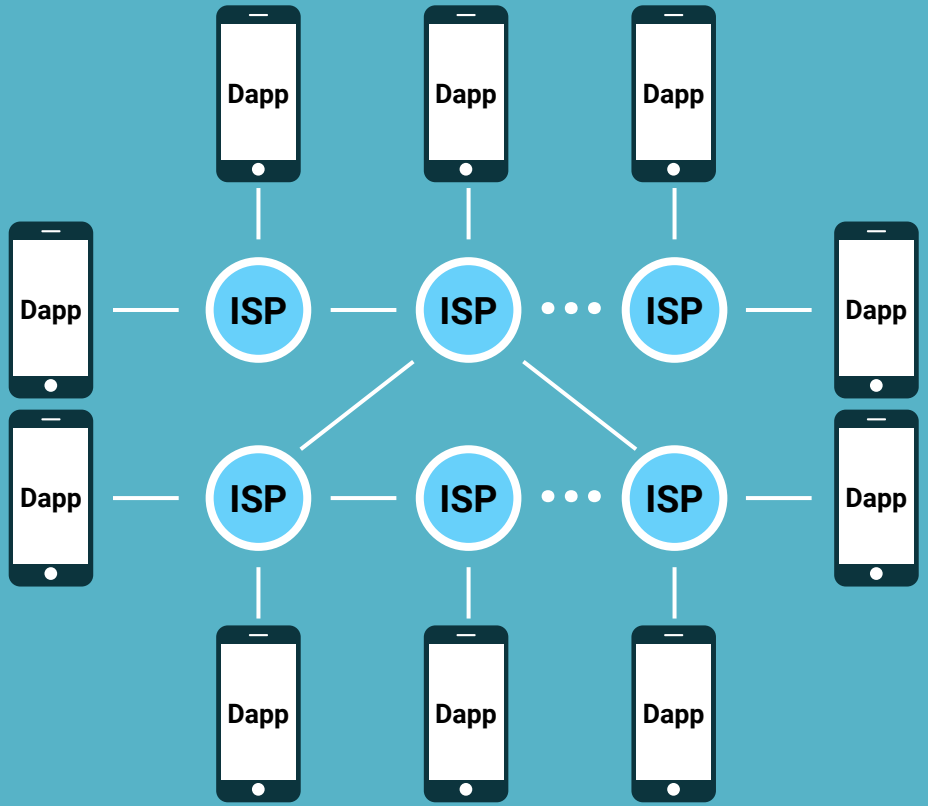
Apps

Run on a centralized server
and use centralized storage.



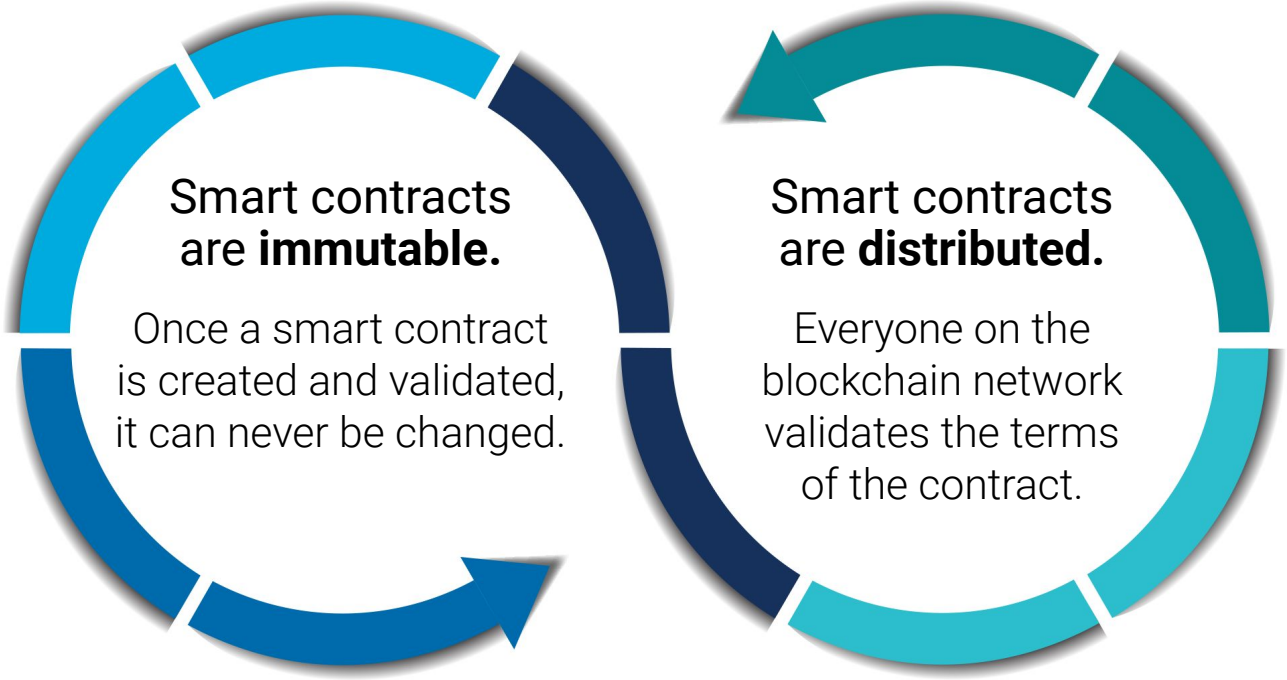
dApps

Run in a decentralized environment
that the blockchain nodes provide.



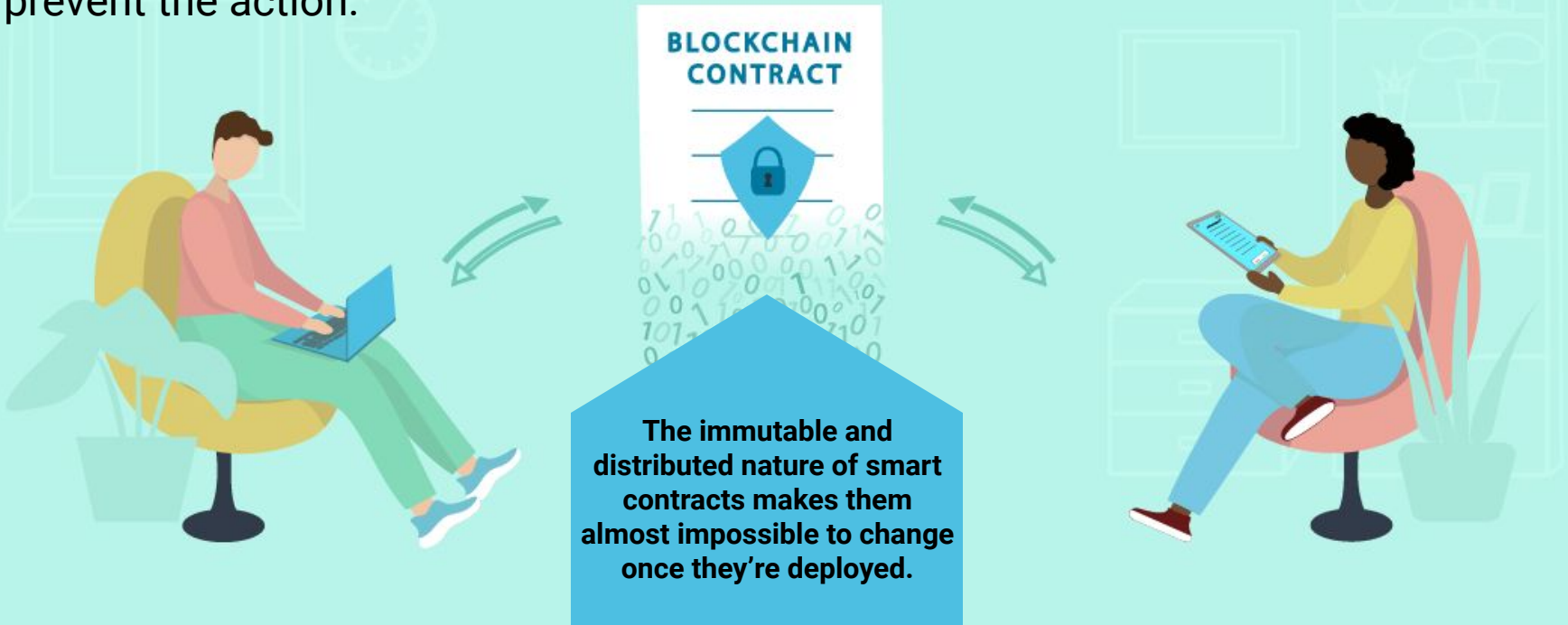
Introduction to Smart Contracts

Smart contracts that exist on a blockchain inherit two valuable properties:



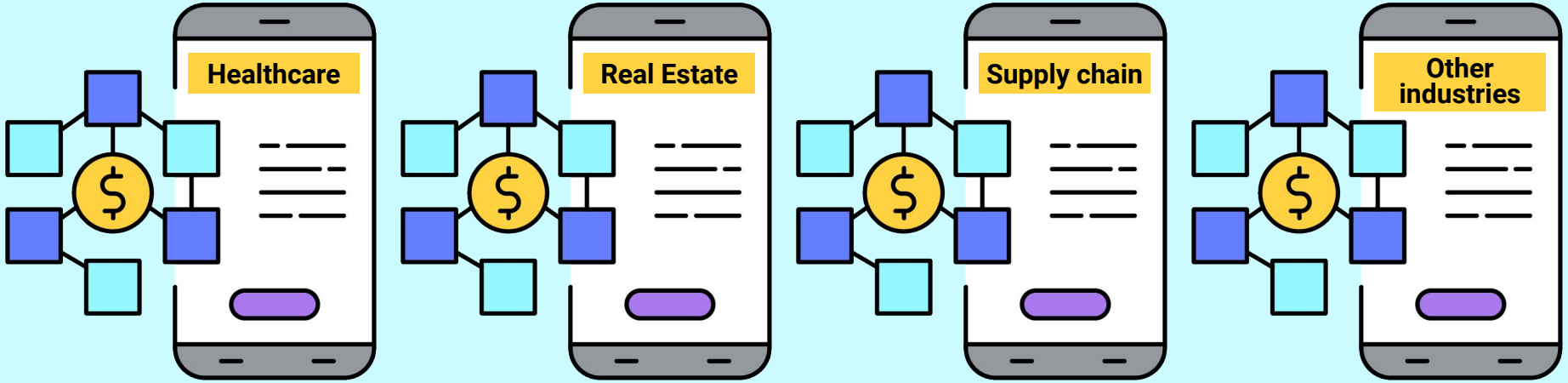
Introduction to Smart Contracts

If one party to the contract or member of the blockchain network tries to override the terms of the contract by trying to trigger an early release of funds, the other network members will recognize the change to the contract as invalid and prevent the action.



Introduction to Smart Contracts

That’s why, in addition to the financial sector extensively using them, people are using smart contracts to solve problems in:



People can even use smart contracts to build decentralized games, voting applications, and marriage contracts.

Introduction to Smart Contracts

Smart contracts need to allow application development and credible transactions that involve assets without a third-party overseer. So, smart contracts rely on the following building blocks:

A robust and reliable blockchain technology

A robust and reliable programming language

An isolated execution environment

Robust encryption mechanisms

An industry-supported blockchain technology

A suitable set of development tools for creating and deploying the smart contracts

These building blocks involve a lot, but as you work with smart contracts, they'll all start to make more sense.

Introduction to Smart Contracts

In the upcoming sections, you'll learn about the differences between Solidity and Python. This will involve how they create, compile, and use the code. Then, you'll build your very first smart contract by using this new language!

vs.

How would you define a smart contract to someone who's new to blockchain technology?

Smart contracts are just computer programs that run on a blockchain.

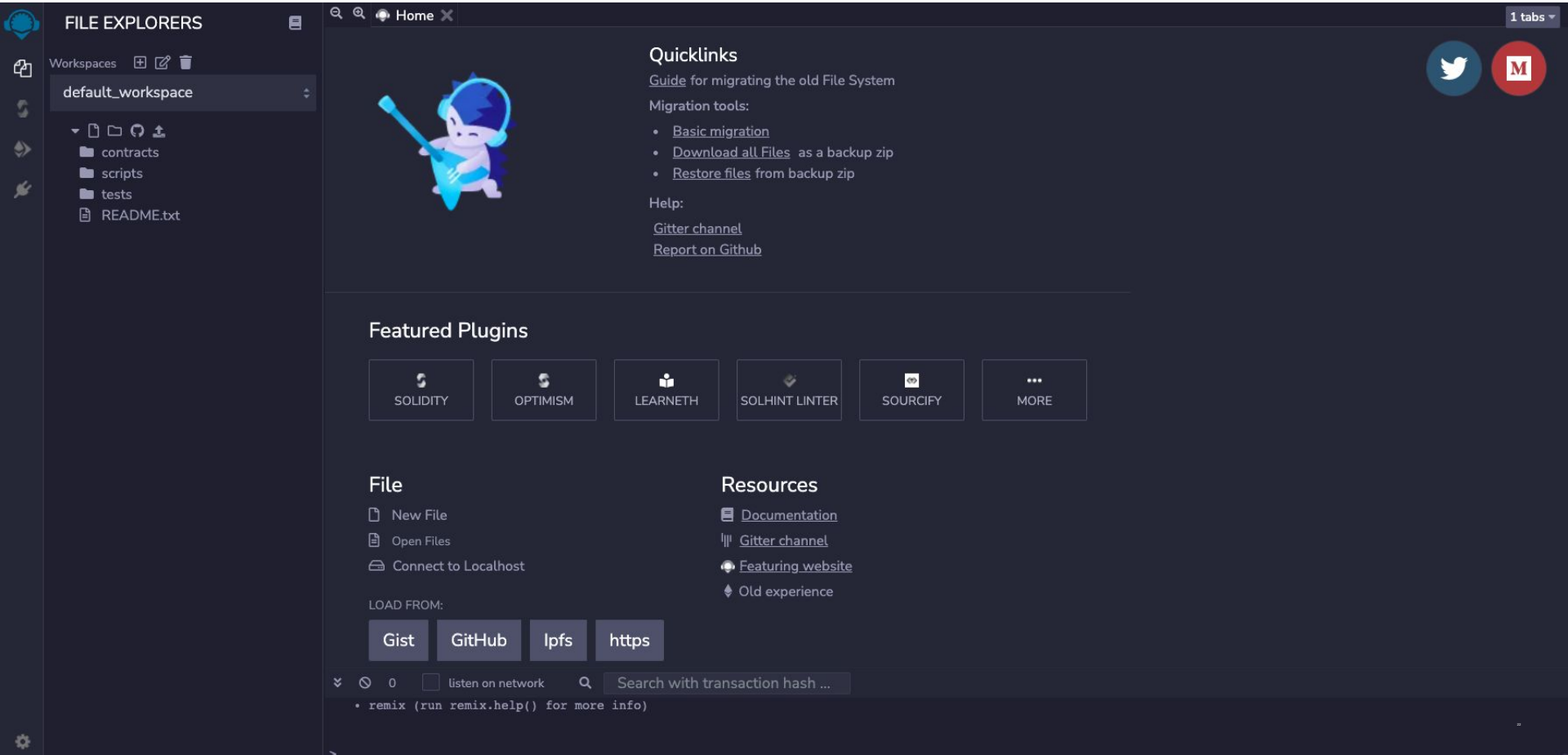
People use them for credible transactions of digital assets without third parties.

Solidity and Remix

Solidity is the programming language that people use for smart contract development on the Ethereum blockchain.

Remix is the integrated development environment (IDE) that we use for Solidity development and deployment.

Remix Integrated Development Environment (IDE)



Solidity and Remix

Solidity will help us develop smart contracts that work on Ethereum-compatible blockchains. These include:

Hyperledger Fabric from the Linux Foundation

HYPERLEDGER

Quorum from Consensys

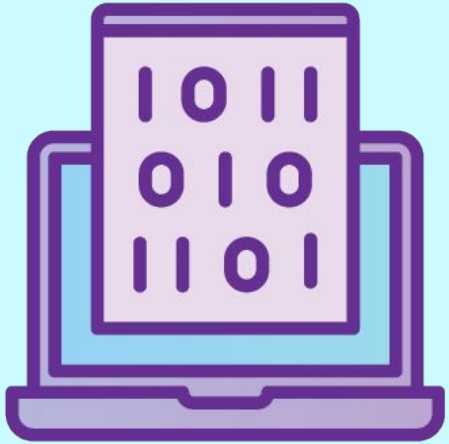
CONSENSYS Quorum

Ethereum Classic

Before getting into the fundamentals of Solidity coding, we need to discuss how to write and run Solidity code.

Solidity and Remix

Solidity code is considered particularly human readable, with characteristics that resemble those of both Python and JavaScript.



When a Solidity smart contract runs, the program first gets compiled.

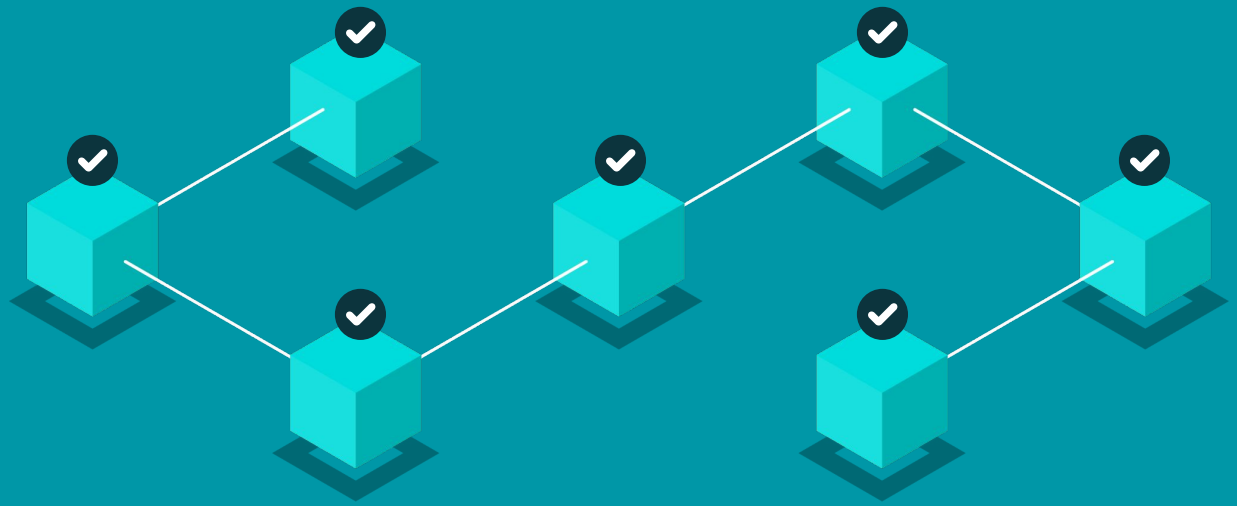
This means that it's converted from human-readable syntax into a computer-friendly version of the code, called bytecode.

Once compiled, the code runs in the EVM.

The **EVM** is a software platform that's embedded in each node of the Ethereum blockchain network. The EVM runs the bytecode that results from running a Solidity smart contract.

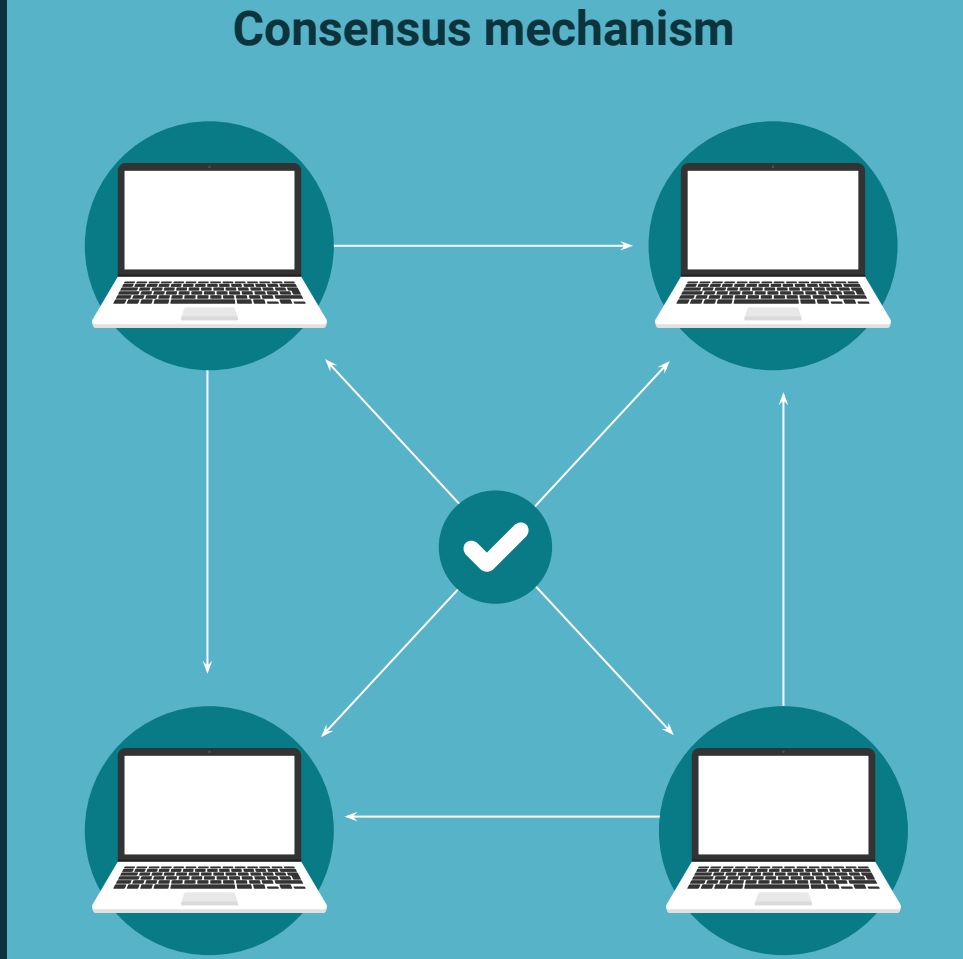
A Key Benefit of Using the EVM

The EVM offers an isolated environment in which Solidity code can run—without accessing the computer network, files, or other resources of the host computer.



This means that by using that same instructions, each decentralized node on the Ethereum blockchain can validate the smart contract. The EVM is thus essential to the consensus mechanism of the Ethereum blockchain.

In general, a **consensus mechanism** allows the nodes in the Ethereum blockchain to work together to both stay secure and reach an agreement on the current state of the blockchain.

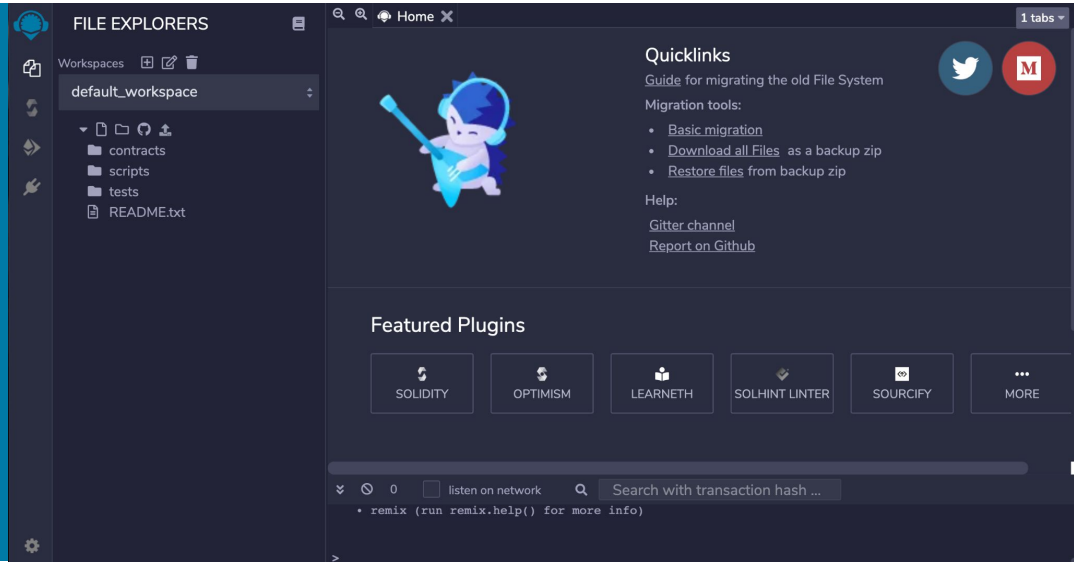


Remix IDE

We use this IDE for Solidity, just like we use an IDE for Python. But instead of using Visual Studio Code or JupyterLab, we'll use the Remix IDE.

The Remix IDE is an open-source application for developing, deploying, and administering smart contracts that run on Ethereum-based blockchains. We can use this IDE for the entire development cycle of smart contracts and as a playground for teaching and learning Ethereum.

The Remix IDE is available in both web and desktop versions. For better compatibility among operating systems, we'll use the web version of the Remix IDE.



Because Remix is an open-source application, the Remix IDE is under constant development, and its user interface often gets updated. So, the interface in the current live version might vary from the slides that appear in this lesson.

https://remix.ethereum.org

Class Slack Channel:

https://remix-project.org/

If you want to learn more about the entire Remix Project
or even join its development community, this is an excellent resource.

Instructor Demonstration

Coding Our First Smart Contract

Solidity syntax differs a bit from Python syntax. That's partly because Solidity is an object-oriented programming language.

Case Styles

Solidity coding styles originated from the PEP 8 — Style Guide for Python Code.

`CapWords`	Capitalize the first letter of each word.
`MixedCase`	Use lowercase for the first word, and capitalize the first letter of each subsequent word.
`camelCase`	Spaces and punctuation are removed, and the first letter of each word is capitalized.
`ALL CAPS`	A unicase style with capital letters only.
`lowercase`	A unicase style with no capital letters.

Solidity is a programming language that's **statically typed**. This means that we need to specify a data type for each Solidity variable.

Data Types

Solidity includes several data types. This image lists the main ones, which you can use for smart contracts:

Data Type	Python Example	Solidity Example
String	`some_string = "Name"`	`string someString = "Name";`
Positive number	`some_number = 123123123`	`unit someNumber = 123123123;`
Negative number	`some_number = -321123`	`int someNumber = -321123;`
Wallet address	`Not applicable`	`address myEthAddress = 0xc3879B456DAA348a16B6524CBC558d2CC984722c;`
Boolean value	`some_condition = True`	`bool someCondition = true;`

Data Types

A variable of type `string` stores a text value.

A variable of type `uint` stores a positive number. The keyword `uint` stands for “unsigned integer.”

A variable of type `int` stores a number. This type of variable can store a positive or a negative integer.

A variable of type `address` stores an Ethereum address. This is a special Solidity data type for storing an Ethereum address in a way that’s computationally more efficient than storing a string.

A variable of type `bool` stores a Boolean value—that is, `true` or `false`.

Questions?

Time to Code

Create a Customer Contract

Suggested Time:

20 minutes

Time's Up! Let's Review.

Questions?

Break

Instructor Demonstration

Solidity Functions

A **buy order** refers to the action of buying a certain amount of financial assets. In this case, we're referring to buying cryptocurrencies.

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

EVM

EVM can store items in three areas:

01

In storage:

Contract variables all reside in storage.

02

In memory:

Temporary values reside in memory.

The EVM clears this memory area between function calls. So, it's less expensive to use than storage.

03

On the stack:

Holds small local variables and argument values.

It's almost free to use but can store only a limited number of values.

A great way to learn more about these storage areas is to read the Storage, Memory and the Stack section of the Solidity documentation.

Access control, in the object-oriented programming paradigm, allows a function to be called from outside the contract—by either users or other contracts.

Questions?

Time to Code

Travel Expenses Contract

Suggested Time:

30 minutes

Time's Up! Let's Review.

Questions?

Instructor Demonstration

Getters and Setters

Getters and Setters

A common practice in object-oriented programming languages is to define functions that only update the value of variables or that only get the current values.

Setter

The `updateTrade` function sets the values of the variables.

This function is called a **setter**.

A setter can set or update the value of any variable.

Getter

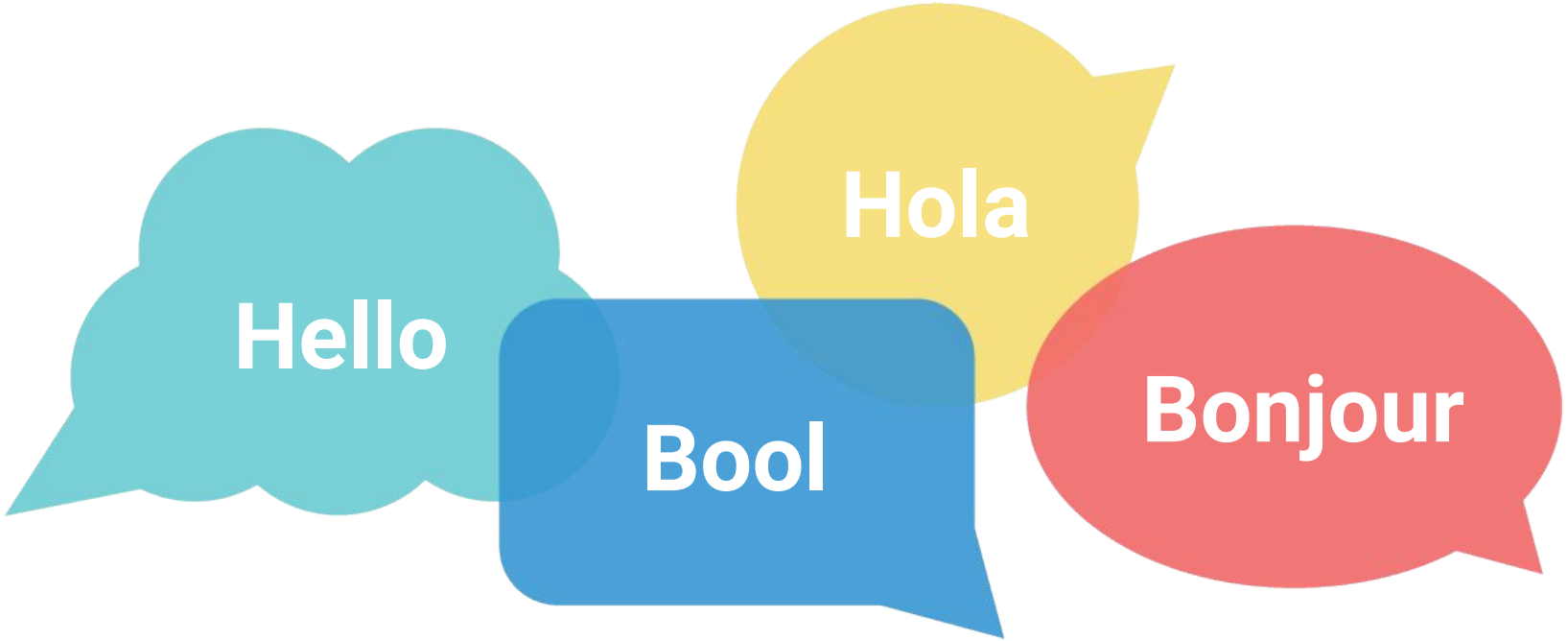
The `getLatestTrade` function gets the current values of the variables.

This function is called a getter.

A getter can get the current value of a variable.

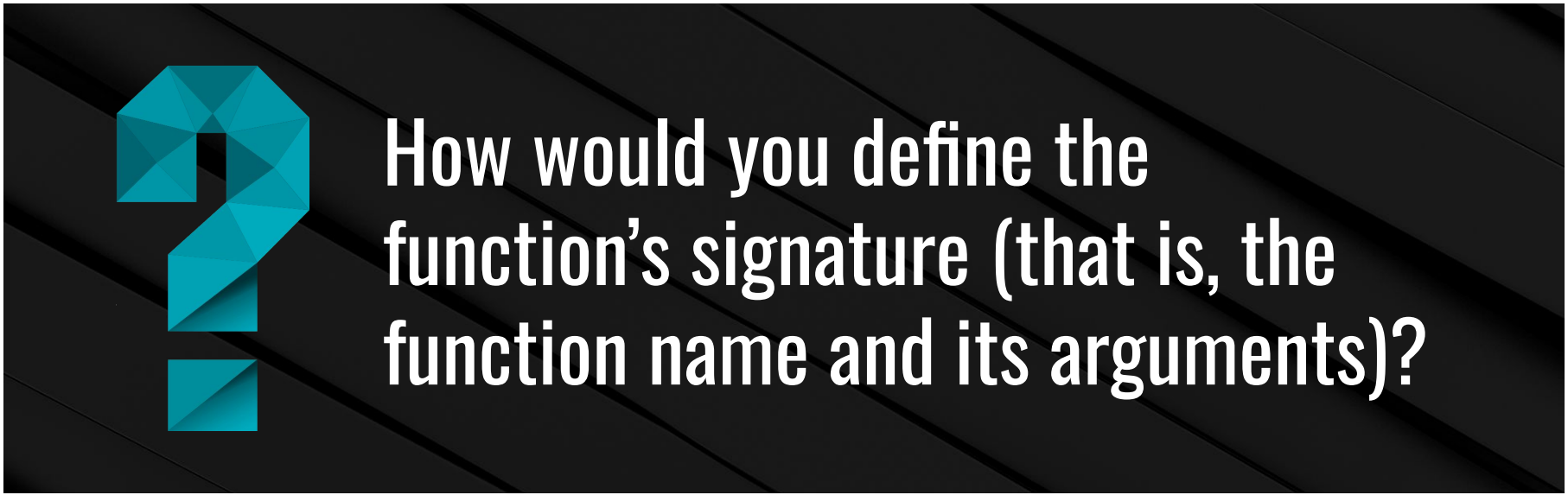
The key to learning is practice.

Learning a new programming language is like learning a new human language.



You'll have the opportunity to practice your new Solidity skills throughout the remainder of this unit.

Suppose that you want to create a smart contract that allows people to transfer funds between two Ethereum addresses. You need a function that sets the transfer amount, the sender, and the recipient.



Function Signature

You might name your function `setTransferInfo` and define its signature as follows:

```
function setTransferInfo(address sender, address recipient, uint amount) public {
  // set user data here
}
```

Time to Code

Adding Getters and Setters

Suggested Time:

20 minutes

Time's Up! Let's Review.

Questions?

The End