COLOMBIA POTENCIA DE LA VIDA

TIC

Lección 2
Smart contracts

M2- Unidad 1

TALENTO TECH

AZ | PROYECTOS EDUCATIVOS

UTP Universidad Tecnológica de Pereira

Tiempo de ejecución: 6 horas

PLANTEAMIENTO DE LA SESIÓN
El desarrollo y funcionamiento de los Smart Contracts está respaldado por una variedad de plataformas blockchain, cada una con sus propias características, ventajas y desventajas. En esta lección, nos sumergiremos en los detalles de estas plataformas para comprender cómo permiten la creación y ejecución de contratos inteligentes, y cómo sus diferencias influyen en el diseño y la implementación de los mismos. Variedad de Plataformas y Protocolos de Consenso Desde Ethereum hasta Binance Smart Chain, y pasando por otras plataformas emergentes, el ecosistema de plataformas para Smart Contracts es vasto y diverso. Cada plataforma utiliza su propio protocolo de consenso, que determina cómo se validan y ejecutan los contratos inteligentes. Ethereum, por ejemplo, utiliza el protocolo de consenso de Prueba de Trabajo (PoW), mientras que Binance Smart Chain utiliza un enfoque de Prueba de Participación Autorizada (PoSA). Estas diferencias en los protocolos de consenso tienen un impacto significativo en aspectos como la seguridad, la escalabilidad y el costo de las transacciones. Lenguajes de Programación y Funcionalidades

Desarrollo teórico de la sesión: 2 horas

Otro aspecto importante a considerar son los lenguajes de programación compatibles y las funcionalidades ofrecidas por cada plataforma. Ethereum es conocido por su compatibilidad con Solidity, un lenguaje específico para contratos inteligentes, mientras que otras plataformas como Tezos ofrecen soporte para múltiples lenguajes de programación, como Michelson y SmartPy. Las funcionalidades también varían entre plataformas, desde la capacidad de crear tokens personalizados hasta el soporte para oráculos externos y la interoperabilidad con otras blockchains. Impacto en el Diseño y la Implementación de Smart Contracts Estas diferencias entre plataformas tienen un impacto significativo en el diseño y la implementación de contratos inteligentes. Los desarrolladores deben considerar cuidadosamente qué plataforma utilizar en función de los requisitos específicos de su aplicación. Por ejemplo, si se prioriza la velocidad y la escalabilidad, Binance Smart Chain puede ser una opción más adecuada, mientras que si se valora la seguridad y la estandarización, Ethereum podría ser la elección preferida. Contribuciones al Crecimiento y la Innovación en Blockchain En última instancia, este amplio espectro de plataformas para Smart Contracts contribuye al crecimiento y la innovación en el ámbito de los contratos inteligentes en blockchain. La competencia entre plataformas impulsa la mejora continua y la adopción más amplia de la tecnología de contratos inteligentes, lo que beneficia a toda la comunidad blockchain y abre nuevas posibilidades para aplicaciones descentralizadas en una variedad de industrias.

Desarrollo de la sesión

El desarrollo y funcionamiento de los Smart Contracts están respaldados por una variedad de plataformas, cada una con sus propias características y ventajas. En esta fase, se explora la evolución y la importancia de los Smart Contracts en el contexto de la tecnología blockchain. Una de las primeras apariciones de los Smart Contracts se remonta al concepto introducido por Nick Szabo en la década de 1990. Szabo definió los Smart Contracts como protocolos informáticos que facilitan, verifican o hacen cumplir la negociación o ejecución de un contrato. Sin embargo, su implementación práctica se vio limitada debido a la falta de una infraestructura adecuada.

Con el surgimiento de blockchain, especialmente con la llegada de Ethereum en 2015, los Smart Contracts se convirtieron en una realidad tangible. Ethereum permitió la ejecución de contratos inteligentes utilizando su lenguaje de programación Turing completo llamado Solidity. El concepto de los Smart Contracts revolucionó la forma en que se llevan a cabo los acuerdos digitales, al permitir la ejecución automática y descentralizada de contratos sin la necesidad de intermediarios. Esto garantiza la transparencia, la seguridad y la inmutabilidad de los acuerdos, lo que ha llevado a su adopción en una amplia gama de aplicaciones y sectores. Plataformas como Ethereum, Binance Smart Chain, EOS y Tron han surgido como líderes en el espacio de los Smart Contracts, ofreciendo entornos de desarrollo robustos y capacidades avanzadas para la creación y ejecución de contratos inteligentes.

Ethereum, en particular, ha sido pionero en este campo, permitiendo la creación de aplicaciones descentralizadas (dApps) que hacen uso extensivo de Smart Contracts para gestionar la lógica empresarial y la interacción entre los usuarios de la plataforma. Además de Ethereum, otras plataformas como Binance Smart Chain ofrecen una alternativa más económica y rápida para el desarrollo y la ejecución de Smart Contracts, lo que ha llevado a una creciente diversificación en el ecosistema de contratos inteligentes.

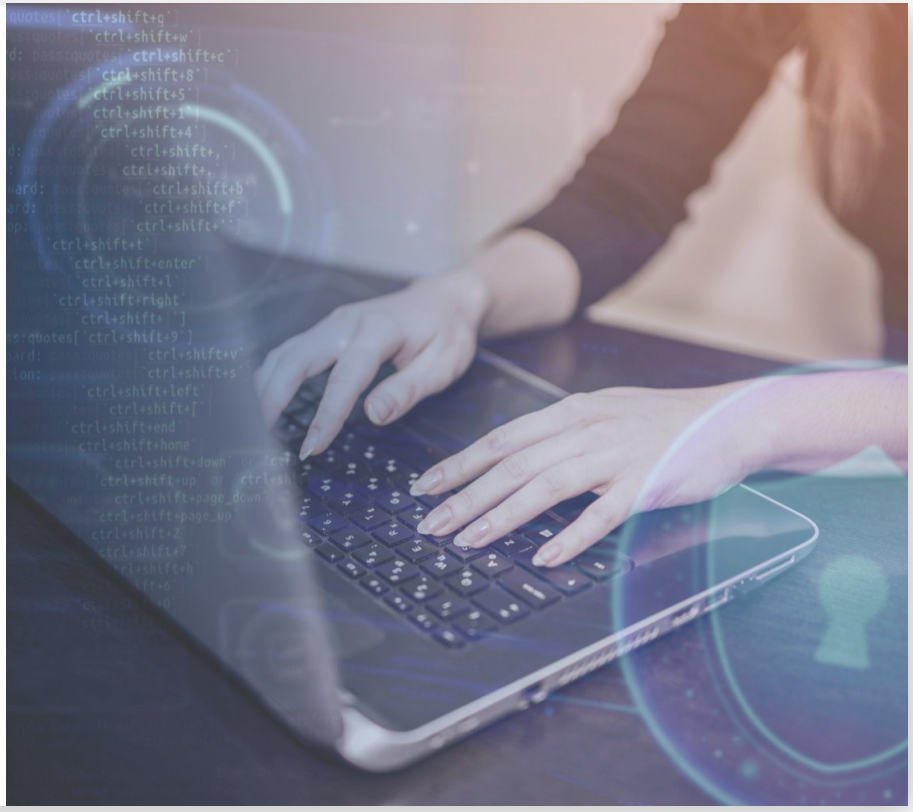
Desarrollo de Smart Contracts

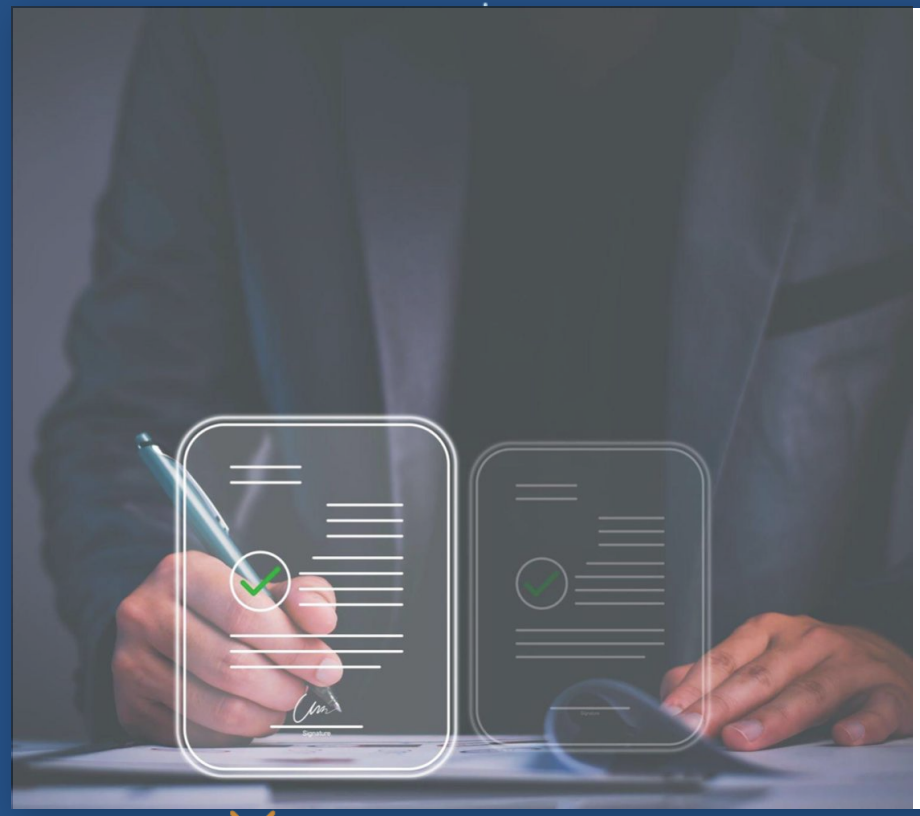
El desarrollo de Smart Contracts implica la codificación de reglas y condiciones específicas en un lenguaje de programación compatible con la plataforma blockchain seleccionada. Este proceso es fundamental para la automatización de acuerdos y transacciones en el entorno descentralizado de la blockchain. Además del lenguaje Solidity, ampliamente utilizado en la plataforma Ethereum, existen otros lenguajes que también se emplean para desarrollar contratos inteligentes en diversas plataformas blockchain.

SOLIDITY Y SU DOMINIO EN ETHEREUM

Solidity

Solidity es el lenguaje de programación más comúnmente asociado con Ethereum y es ampliamente utilizado para desarrollar Smart Contracts en esta plataforma. Este lenguaje ofrece una amplia gama de características y funcionalidades específicas para la creación de contratos inteligentes.





Características y Funcionalidades de Solidity

Solidity está diseñado para facilitar la creación de contratos inteligentes mediante una sintaxis familiar para los desarrolladores de software. Ofrece soporte para tipos de datos, estructuras de control y funciones, lo que permite a los desarrolladores implementar lógica empresarial compleja en la blockchain de Ethereum. Además, Solidity proporciona características avanzadas como la herencia de contratos, la sobrecarga de funciones y la creación de interfaces, que permiten una programación modular y reutilizable.

Alternativas a Solidity

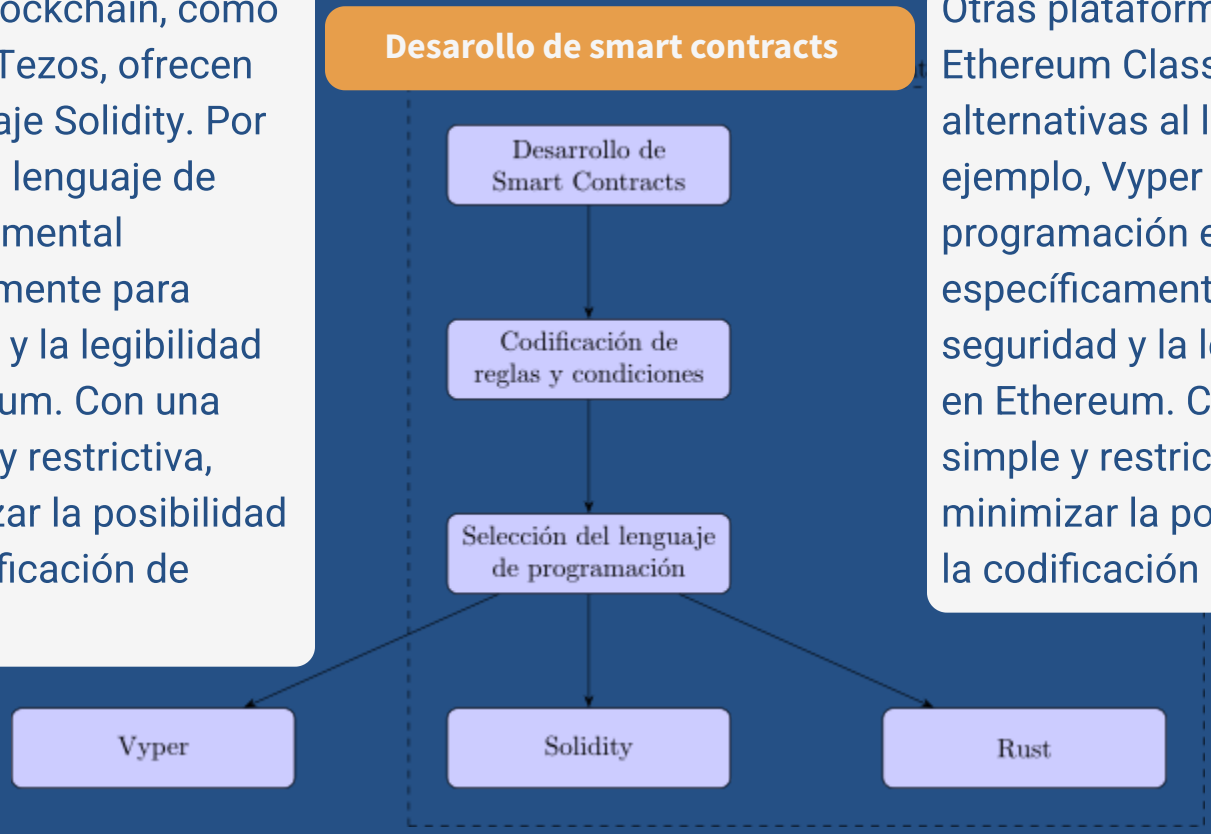
A pesar de su popularidad y amplia adopción, Solidity no es la única opción disponible para los desarrolladores de Ethereum. Otros lenguajes de programación como Vyper y Rust también están ganando terreno en el desarrollo de Smart Contracts. Vyper, por ejemplo, se ha desarrollado específicamente para mejorar la seguridad y la legibilidad del código en Ethereum, mientras que Rust ofrece un enfoque más generalizado con énfasis en la seguridad y el rendimiento.

A medida que el ecosistema de Ethereum evoluciona, los desarrolladores tienen la oportunidad de explorar una variedad de opciones de lenguaje de programación para sus proyectos de Smart Contracts. La elección del lenguaje adecuado puede depender de factores como la seguridad, la legibilidad del código y la familiaridad del desarrollador con el lenguaje en cuestión. Además, es importante considerar la comunidad de desarrolladores y el soporte disponible para cada lenguaje.

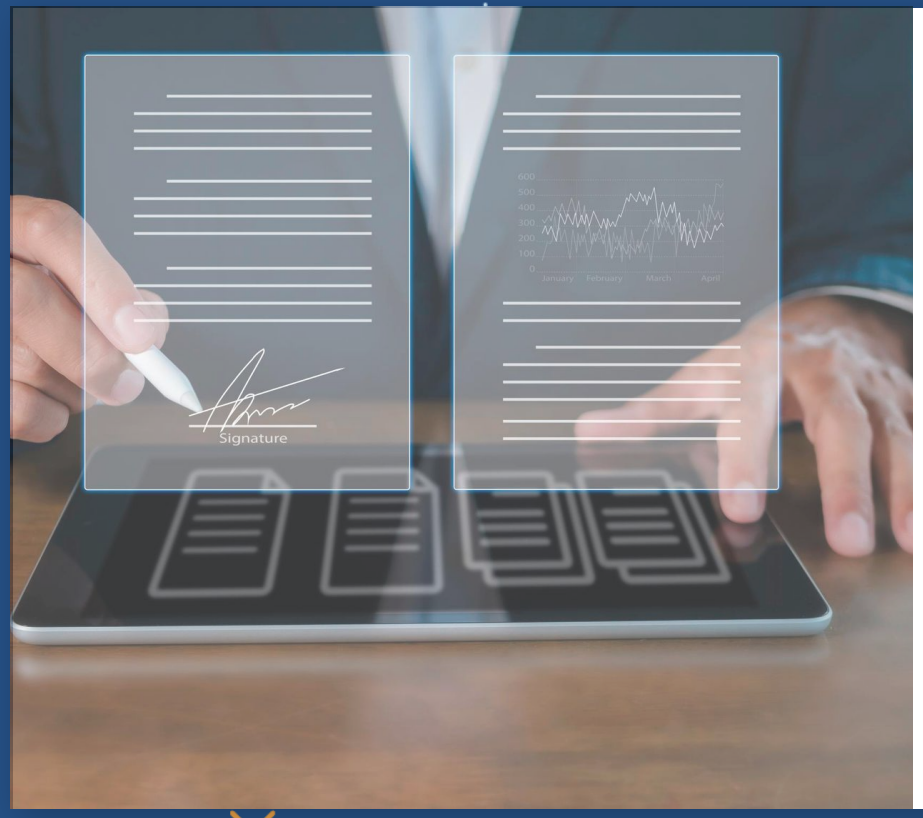
Con una variedad de lenguajes de programación disponibles, los desarrolladores tienen la flexibilidad de seleccionar la opción que mejor se adapte a las necesidades específicas de su proyecto. Ya sea utilizando Solidity, Vyper, Rust u otra opción, la elección del lenguaje de programación adecuado es fundamental para el desarrollo exitoso de Smart Contracts en la plataforma Ethereum.

Lista de muchos elementos

Otras plataformas blockchain, como Ethereum Classic, y Tezos, ofrecen alternativas al lenguaje Solidity. Por ejemplo, Vyper es un lenguaje de programación experimental diseñado específicamente para mejorar la seguridad y la legibilidad del código en Ethereum. Con una sintaxis más simple y restrictiva, Vyper busca minimizar la posibilidad de errores en la codificación de Smart Contracts.



Otras plataformas blockchain, como Ethereum Classic, y Tezos, ofrecen alternativas al lenguaje Solidity. Por ejemplo, Vyper es un lenguaje de programación experimental diseñado específicamente para mejorar la seguridad y la legibilidad del código en Ethereum. Con una sintaxis más simple y restrictiva, Vyper busca minimizar la posibilidad de errores en la codificación de Smart Contracts.



Ejecución y Validación

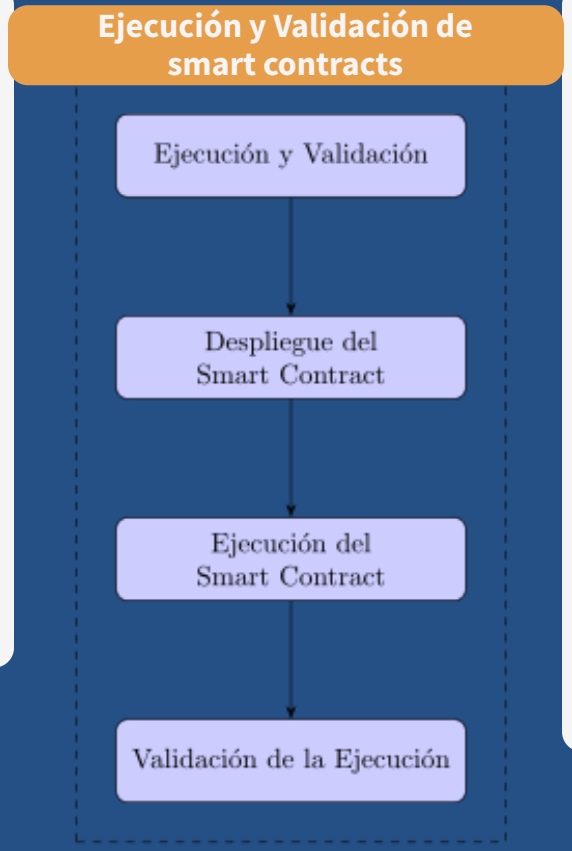
Una vez que un Smart Contract ha sido desplegado en la blockchain, se convierte en una entidad activa que puede ser ejecutada por usuarios y otros contratos. La ejecución de un contrato implica la activación de sus funciones y la modificación del estado de la blockchain de acuerdo con las condiciones establecidas en el contrato.

Ejecución de Funciones y Actualización del Estado

Cuando un usuario o un contrato invoca una función de un Smart Contract, se inicia el proceso de ejecución. Las funciones pueden realizar una variedad de acciones, como transferir tokens, actualizar datos en la blockchain o realizar cálculos complejos. Cada vez que se ejecuta una función, el estado de la blockchain se actualiza de acuerdo con las operaciones especificadas en el contrato.

Validación de la Ejecución

La validación de la ejecución de un Smart Contract es crucial para garantizar la integridad y la inmutabilidad de los contratos en la blockchain. Esta validación se lleva a cabo mediante la verificación de las transacciones por parte de los nodos de la red. Los nodos son responsables de validar las transacciones y asegurarse de que cumplan con las reglas y condiciones establecidas en los contratos inteligentes.



Los nodos de la red trabajan en conjunto para verificar y validar las transacciones, utilizando algoritmos de consenso para garantizar que todas las copias de la blockchain sean idénticas y que no se produzcan alteraciones no autorizadas en los contratos inteligentes. Esta colaboración entre nodos garantiza la integridad y la inmutabilidad de los contratos, lo que significa que una vez que se ha ejecutado un Smart Contract y se ha registrado en la blockchain, su resultado no puede ser alterado ni revertido.

Interoperabilidad y Estándares

Con el crecimiento constante del ecosistema blockchain, se ha vuelto cada vez más evidente la necesidad de establecer estándares y protocolos que faciliten la interoperabilidad entre diferentes plataformas y contratos inteligentes. La interoperabilidad es fundamental para permitir que las diferentes partes del ecosistema interactúen de manera fluida y eficiente, lo que fomenta la colaboración y la innovación en el espacio blockchain.

Estándares Propuestos

Para abordar esta necesidad, se han propuesto varios estándares que definen interfaces comunes y protocolos de comunicación entre plataformas y contratos inteligentes. Uno de los estándares más conocidos es ERC (Ethereum Request for Comments), que establece interfaces comunes para tokens y contratos en la red Ethereum. Estos estándares son ampliamente utilizados en Ethereum y facilitan la interoperabilidad entre diferentes aplicaciones y servicios dentro de la plataforma.

BEP en Binance Smart Chain

Además de ERC, otras plataformas también han desarrollado sus propios estándares para facilitar la interoperabilidad. Por ejemplo, Binance Smart Chain utiliza el estándar BEP (Binance Smart Chain Extension Proposal) para definir interfaces comunes y protocolos de comunicación entre contratos inteligentes en su red. BEP permite la compatibilidad entre diferentes contratos inteligentes en Binance Smart Chain, lo que facilita la integración de aplicaciones y servicios en la plataforma.

Estos estándares son esenciales para facilitar la integración y la colaboración en el ecosistema blockchain. Al establecer interfaces comunes y protocolos de comunicación, los estándares permiten que diferentes partes del ecosistema se comuniquen de manera eficiente, lo que facilita la interoperabilidad entre plataformas y contratos inteligentes. Esto abre nuevas oportunidades para la creación de aplicaciones y servicios innovadores que pueden aprovechar las capacidades de múltiples plataformas blockchain.

La interoperabilidad y los estándares en el ecosistema blockchain son fundamentales para impulsar la innovación y el crecimiento en el espacio blockchain. Al fomentar la colaboración y la integración entre diferentes partes del ecosistema, los estándares permiten que se desarrollen nuevas aplicaciones y servicios que pueden aprovechar las capacidades de múltiples plataformas. Esto promueve un mayor desarrollo y adopción de tecnologías blockchain, lo que beneficia a toda la comunidad blockchain y abre nuevas oportunidades para la innovación en diversos sectores.

Seguridad y Auditoría

Con el rápido crecimiento del ecosistema blockchain, la proliferación de diferentes plataformas y la diversificación de contratos inteligentes, surge la necesidad imperativa de establecer estándares que faciliten la interoperabilidad entre estos componentes. La interoperabilidad se convierte en un facilitador crítico para la integración fluida de Smart Contracts en diversas plataformas, lo que amplía considerablemente su utilidad y alcance en el mundo digital.

Enlace entre Plataformas

Los estándares, como el ERC (Ethereum Request for Comments) en Ethereum, han surgido como puentes esenciales entre las diversas plataformas blockchain. Estos estándares definen interfaces comunes y protocolos de comunicación que permiten que los Smart Contracts funcionen de manera uniforme en diferentes entornos blockchain. Además del ERC, cada plataforma puede tener sus propios estándares, como BEP (Binance Smart Chain Extension Proposal) en Binance Smart Chain, que desempeñan un papel similar en su ecosistema respectivo.

Los estándares interoperables promueven la integración y la colaboración al permitir que los desarrolladores creen Smart Contracts que sean compatibles con múltiples plataformas. Esto abre un abanico de posibilidades, ya que los Smart Contracts pueden ser utilizados en una variedad de aplicaciones y servicios que abarcan diferentes cadenas de bloques. Además, la interoperabilidad facilita la colaboración entre proyectos y comunidades, fomentando un ambiente de innovación compartida en el espacio blockchain.

La adopción y la innovación en el espacio blockchain se ven impulsadas por la interoperabilidad y los estándares. Al eliminar las barreras de entrada y promover la compatibilidad entre diferentes plataformas, se abren nuevas oportunidades para el desarrollo de aplicaciones y servicios innovadores. Los estándares interoperables fomentan un ecosistema más dinámico y colaborativo, lo que a su vez impulsa la adopción generalizada del blockchain y cataliza la evolución continua de la tecnología.

A medida que el ecosistema blockchain evoluciona, es crucial mantener una colaboración continua en el desarrollo y la implementación de estándares interoperables. Esta cooperación garantiza que el blockchain siga siendo un medio flexible y adaptable que pueda satisfacer las necesidades cambiantes de la industria y la sociedad en general. La colaboración entre plataformas, proyectos y comunidades es fundamental para mantener un ecosistema blockchain próspero y en constante crecimiento.

Gobernanza y Actualizaciones:

Con el constante avance y la evolución de las necesidades y requisitos en el ecosistema blockchain, es crucial contar con mecanismos de gobernanza que permitan la actualización y mejora continua de los Smart Contracts. La gobernanza efectiva no solo asegura la viabilidad a largo plazo de las aplicaciones y servicios basados en blockchain, sino que también promueve la confianza y la participación de la comunidad en la toma de decisiones.

Gobernanza Descentralizada

Algunas plataformas blockchain optan por sistemas de gobernanza descentralizada, donde los titulares de tokens o participantes de la red tienen la oportunidad de participar activamente en el proceso de toma de decisiones. Estos sistemas suelen implementar mecanismos de votación, donde los titulares de tokens pueden emitir votos sobre propuestas de actualización o cambios en los contratos inteligentes.

Los mecanismos de gobernanza descentralizada fomentan la adaptabilidad y la innovación al permitir que la comunidad ajuste y mejore continuamente los Smart Contracts en función de las necesidades cambiantes del ecosistema. Esto puede incluir la introducción de nuevas características, la corrección de errores o la implementación de mejoras de rendimiento para optimizar el funcionamiento de las aplicaciones y servicios blockchain.

La transparencia y la participación son aspectos fundamentales de la gobernanza descentralizada en el blockchain. Al permitir que los titulares de tokens participen en el proceso de toma de decisiones, se promueve la transparencia y se garantiza que las actualizaciones y cambios en los contratos inteligentes reflejen los intereses y la voluntad colectiva de la comunidad.

Sostenibilidad a Largo Plazo

La gobernanza descentralizada contribuye a la sostenibilidad a largo plazo del ecosistema blockchain al asegurar que las actualizaciones y mejoras se realicen de manera colaborativa y consensuada. Esto ayuda a prevenir bifurcaciones y conflictos internos, lo que a su vez promueve la estabilidad y la confianza en el blockchain como una infraestructura confiable para aplicaciones y servicios digitales.

La evolución del blockchain depende en gran medida de una cooperación continua entre todos los actores del ecosistema. La gobernanza descentralizada facilita esta cooperación al proporcionar un marco transparente y participativo para la toma de decisiones. Al trabajar juntos para mejorar y adaptar los Smart Contracts, la comunidad blockchain avanza hacia un futuro más resiliente, innovador y sostenible.

Actividad:

Algunas plataformas blockchain optan por sistemas de gobernanza descentralizada, donde los titulares de tokens o participantes de la red tienen la oportunidad de participar activamente en el proceso de toma de decisiones. Estos sistemas suelen implementar mecanismos de votación, donde los titulares de tokens pueden emitir votos sobre propuestas de actualización o cambios en los contratos inteligentes.

1. **Introducción a Smart Contracts y Blockchain:** Comience la sesión proporcionando una breve introducción sobre Smart Contracts y su relación con la tecnología blockchain(Ver lección anterior). Destaque su importancia en la automatización y descentralización de acuerdos digitales.

Para comenzar a comprender el desarrollo de contratos inteligentes en Solidity, es fundamental conocer las generalidades de este lenguaje de programación. Iniciar con un ejemplo básico, como el famoso "Hola Mundo", proporciona una introducción esencial al código Solidity. En un archivo *.sol*, un simple contrato que imprime "Hola Mundo" puede servir como punto de partida para explorar los conceptos básicos de Solidity. Después de escribir el contrato en un editor de texto, como Visual Studio Code, el siguiente paso es compilarlo utilizando el compilador de Solidity (solc). La compilación genera dos archivos: uno que contiene el bytecode del contrato y otro que contiene la ABI (Interfaz de Lenguaje Binario). Estos archivos son fundamentales para el despliegue y la interacción con el contrato inteligente en una red blockchain. Este tipo de ejercicio no solo familiariza a los desarrolladores con la estructura del código en Solidity, sino que también establece una base sólida para abordar conceptos más avanzados en el desarrollo de contratos inteligentes.

```
// SPDX-License-Identifier: MIT
 pragma solidity ^0.8.0;
 contract MiContrato
{ string public mensaje = "Hola
Mundo!";
}
```

Compilación de Contratos Inteligentes en Solidity

En Windows:

Instalar Solidity: Puedes instalar el compilador de Solidity en Linux utilizando los paquetes disponibles para tu distribución específica. Por ejemplo, en Ubuntu, puedes agregar el repositorio PPA de Ethereum y luego instalar Solidity usando el gestor de paquetes apt.

Escribir el Contrato Inteligente: Utiliza un editor de texto, como Nano o Visual Studio Code, para escribir el código de tu contrato inteligente en un archivo con extensión .sol. Por ejemplo, puedes crear un archivo llamado MiContrato.sol.

Isolc MiContrato.sol --bin --abi --optimize -o ./output
Este comando generará dos archivos en el directorio output: MiContrato.bin (el bytecode del contrato) y MiContrato.abi (la ABI del contrato).

```
// SPDX-License-Identifier: MIT
 pragma solidity ^0.8.0;
 contract MiContrato
{ string public mensaje = "Hola
Mundo!";
}
```

Compilación de Contratos Inteligentes en Solidity

En Linux:

Instalar Solidity: Puedes instalar el compilador de Solidity en Linux utilizando los paquetes disponibles para tu distribución específica. Por ejemplo, en Ubuntu, puedes agregar el repositorio PPA de Ethereum y luego instalar Solidity usando el gestor de paquetes apt.

Escribir el Contrato Inteligente: Utiliza un editor de texto, como Nano o Visual Studio Code, para escribir el código de tu contrato inteligente en un archivo con extensión .sol. Por ejemplo, puedes crear un archivo llamado MiContrato.sol.

Compilar el Contrato: Abre una terminal y navega hasta el directorio que contiene tu archivo .sol. Luego, ejecuta el siguiente comando para compilar tu contrato:
solc MiContrato.sol --bin --abi --optimize -o ./output

```
// SPDX-License-Identifier: MIT
 pragma solidity ^0.8.0;
 contract MiContrato
{ string public mensaje = "Hola
Mundo!";
}
```

Compilación de Contratos Inteligentes en Solidity

2. Práctica de Desarrollo: Divida a los estudiantes en grupos pequeños y proporcione a cada grupo acceso a una plataforma de desarrollo de contratos inteligentes, como Remix IDE para Ethereum. Pídales que trabajen juntos para crear un Smart Contract simple que simule un acuerdo básico, como una transferencia de activos digitales.

A continuación se presenta el código del Smart Contract que será utilizado en la práctica de desarrollo. Este contrato simula un acuerdo básico de transferencia de activos digitales y servirá como base para que los grupos trabajen juntos en la creación y prueba de contratos inteligentes.

+ info

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;
contract TransferenciaActivos {
 struct Parte {
address direccion;
string nombre;
 }
 struct Activo {
uint256 cantidad;
string nombre;
 }
 mapping(address => Activo) public activos;
 event Transferencia(Parte _comprador, Parte _vendedor, Activo _activo);
 function transferirActivo(Parte memory _comprador, Parte memory _vendedor, Activo memory _activo) public {
require(_activo.cantidad > 0, "Cantidad del activo debe ser mayor a 0");
require(activos[_vendedor.direccion].cantidad >= _activo.cantidad, "Vendedor no tiene suficientes activos");
activos[_vendedor.direccion].cantidad -= _activo.cantidad;
activos[_comprador.direccion].cantidad += _activo.cantidad;
emit Transferencia(_comprador, _vendedor, _activo);
 }
}
```

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;
contract TransferenciaActivos {
 struct Parte {
address direccion;
string nombre;
 }
 struct Activo {
uint256 cantidad;
string nombre;
 }
 mapping(address => Activo) public activos;
 event Transferencia(Parte _comprador, Parte _vendedor, Activo _activo);
 function transferirActivo(Parte memory _comprador, Parte memory _vendedor, Activo memory _activo) public {
require(_activo.cantidad > 0, "Cantidad del activo debe ser mayor a 0");
require(activos[_vendedor.direccion].cantidad >= _activo.cantidad, "Vendedor no tiene suficientes activos");
activos[_vendedor.direccion].cantidad -= _activo.cantidad;
activos[_comprador.direccion].cantidad += _activo.cantidad;
emit Transferencia(_comprador, _vendedor, _activo);
 }
}
```

3. Despliegue y Ejecución: Después de que los grupos hayan desarrollado sus Smart Contracts, guíelos a través del proceso de despliegue en una red de prueba, como Ropsten para Ethereum. Instruya a los estudiantes sobre cómo interactuar con los contratos inteligentes utilizando herramientas como MetaMask y Remix IDE. Anime a los grupos a ejecutar transacciones de prueba y a observar cómo se actualiza el estado de la blockchain en respuesta a las acciones realizadas en los contratos inteligentes.

+ info

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;
contract TransferenciaActivos {
 struct Parte {
address direccion;
string nombre;
 }
 struct Activo {
uint256 cantidad;
string nombre;
 }
 mapping(address => Activo) public activos;
 event Transferencia(Parte _comprador, Parte _vendedor, Activo _activo);
 function transferirActivo(Parte memory _comprador, Parte memory _vendedor, Activo memory _activo) public {
require(_activo.cantidad > 0, "Cantidad del activo debe ser mayor a 0");
require(activos[_vendedor.direccion].cantidad >= _activo.cantidad, "Vendedor no tiene suficientes activos");
activos[_vendedor.direccion].cantidad -= _activo.cantidad;
activos[_comprador.direccion].cantidad += _activo.cantidad;
emit Transferencia(_comprador, _vendedor, _activo);
 }
}
```

Después de que los grupos hayan desarrollado sus Smart Contracts, es el momento de llevar a cabo el proceso de despliegue en una red de prueba, como Ropsten para Ethereum. Se les instruirá sobre cómo interactuar con los contratos inteligentes utilizando herramientas como MetaMask y Remix IDE. Se anima a los grupos a ejecutar transacciones de prueba y a observar cómo se actualiza el estado de la blockchain en respuesta a las acciones realizadas en los contratos inteligentes.

Para facilitar este proceso, se proporciona a continuación una guía paso a paso:

1.IDespliegue del Contrato Inteligente en Ropsten:

- Utilizando Remix IDE, conecta MetaMask a la red de prueba Ropsten.
- Compila el Smart Contract desarrollado por tu grupo.
- Despliega el contrato en la red Ropsten utilizando la cuenta de MetaMask.

2.Interacción con el Contrato Inteligente:

- Utilizando Remix IDE o una interfaz de usuario, como la de Etherscan, obtén la dirección del contrato desplegado.
- Utiliza MetaMask para enviar transacciones de prueba al contrato inteligente.
- Observa cómo se actualiza el estado de la blockchain en respuesta a las transacciones ejecutadas.

```
// SPDX-License-Identifier: MIT
 pragma solidity ^0.8.0;
 contract MiContrato
{ string public mensaje = "Hola
Mundo!";
}
```

Compilación de Contratos Inteligentes en Solidity

Exploración de Casos de Uso y Aplicaciones

1. Discusión sobre Casos de Uso:
Facilite una discusión sobre casos de uso potenciales para Smart Contracts en diversos sectores, como finanzas, logística, salud, etc. Anime a los estudiantes a compartir ideas sobre cómo podrían implementar Smart Contracts en escenarios del mundo real y cómo podrían beneficiar a las partes involucradas.

2. Reflexión y Conclusiones:
Concluya la actividad con una sesión de reflexión en la que los estudiantes compartan sus experiencias y aprendizajes sobre el desarrollo y ejecución de Smart Contracts. Fomente la discusión sobre los desafíos y oportunidades asociados con la implementación de esta tecnología, así como sobre posibles mejoras o innovaciones futuras.

```
// SPDX-License-Identifier: MIT
 pragma solidity ^0.8.0;
 contract MiContrato
{ string public mensaje = "Hola
Mundo!";
}
```

Compilación de Contratos Inteligentes en Solidity

Ejercicios en Clase

Durante esta sección, los estudiantes participarán en una serie de ejercicios diseñados para ayudarles a consolidar los conceptos aprendidos sobre Smart Contracts y aplicarlos en situaciones prácticas.

1. **Ejercicio de Desarrollo:** Proporcione a los estudiantes una serie de escenarios hipotéticos que requieren la implementación de Smart Contracts para resolver problemas específicos. Pídales que trabajen individualmente o en parejas para diseñar y desarrollar Smart Contracts que cumplan con los requisitos de cada escenario.

2. **Simulación de Ejecución:** Una vez que los estudiantes hayan desarrollado sus Smart Contracts, organice una sesión de simulación de ejecución donde puedan probar sus contratos en un entorno de prueba. Anime a los estudiantes a realizar transacciones simuladas y a observar cómo se comportan sus contratos en respuesta a diferentes condiciones.

3. **Análisis de Casos:** Después de la simulación de ejecución, anime a los estudiantes a analizar y discutir los resultados obtenidos. Pídales que reflexionen sobre la eficacia de sus Smart Contracts para resolver los problemas planteados en los escenarios y que identifiquen posibles mejoras o ajustes que podrían realizar en sus contratos.