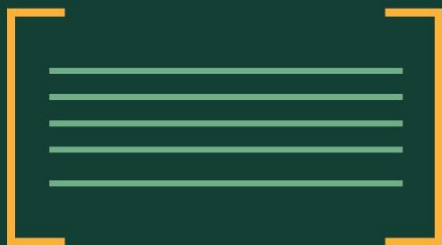


# Curso de **Introducción a Solidity**

---

Sebastián  
Pérez





# Por qué debes aprender Solidity

**Bienvenida al curso**

# Sebastián Pérez



@sebaleoperez  
@blockacademycl



# Temario

- Por qué debes aprender Solidity.
- Introducción a la programación con Solidity.
- Funciones.
- Gestionando costos.
- Conceptos avanzados.
- Cómo continuar mi camino como desarrollador de Blockchain.

# **La importancia de Web3 y Solidity**



# Web3

Así denominamos a la evolución de las aplicaciones que funcionan sobre internet, integrando tecnologías como big data e inteligencia artificial, para lograr escenarios como las redes sociales.



# Blockchain

La aparición de esta tecnología juega un rol clave en la evolución de la web3, ya que agrega una característica innovadora:

**La descentralización.**






# Solidity

No existe una sola plataforma de Blockchain, pero en la diversidad se destacan aquellas que agregan la capacidad de ejecutar código.

Solidity es un lenguaje especialmente creado para el diseño de este código conocido como **contratos inteligentes**.



# Introducción a la programación en Solidity

Contratos, variables y estructuras



# **Estructura de un contrato inteligente**



# Pragma

Todos los contratos comienzan con la **sentencia pragma** el cual señala la versión de Solidity a utilizar.

Puede hacerse referencia a una versión puntual o a un rango de versiones.



# Contract

Para comenzar la declaración del contrato debemos utilizar el keyword **contract**.

Todo lo que se encuentre en el entorno de este keyword será parte del mismo contrato.



# Constructor

Al igual que en la programación orientada a objetos, cada contrato cuenta con una **función opcional constructor** la cual se ejecuta por única vez cuando se crea el contrato.

# **Variables y tipos de datos**



# Números enteros

- **int**

Números enteros. Pueden opcionalmente indicar su tamaño en bits.

- **uint**

Números enteros **sin signo**. Pueden opcionalmente indicar su tamaño en bits.



# Números decimales





# Tipos de datos

- **bool**

Valor booleano, representa verdadero/falso.

- **address**

Almacenan direcciones de Ethereum. Sirve para relacionar cuentas y contratos.



# Tipos de datos

- **string**

Representa un texto en forma de cadena de caracteres.

- **bytes**

Representa una cadena de bytes sin un formato específico.



# Tipos de variables

- **local variables**

Son variables que se alojan de forma temporal para utilizarse dentro del ámbito de una función.

- **state variables**

Son variables que se almacenan en el contrato y que mantienen su valor aun después de finalizada la ejecución de la función.



# Variables globales

- **msg**

Valores relacionados con la configuración del mensaje.

- **tx**

Valores relacionados con la transacción actual.

- **block**

Valores relacionados con el bloque actual.

# **Structs y arrays**



# Struct

Tipo de dato estructurado que nos permite **almacenar información de distintos tipos en un solo esquema.**



# Array

Es una **cadena de elementos del mismo tipo de dato que se almacenan de forma consecutiva** y por lo tanto se puede acceder a un elemento según su posición dentro de la misma.

Se dividen en:

- Arrays estáticos
- Arrays dinámicos



# Mappings y enums



# Mapping

Es un tipo de almacenamiento de un valor identificado por una clave la cual puede ser de un tipo de dato diferente.

Permite acceder al valor directamente desde su clave sin necesidad de iterar todos los valores almacenados.



# Enum

Representa una serie de valores creados por el usuario, los cuales van a ser representados externamente por un nombre, pero internamente por un valor entero.

Son utilizados principalmente para manejar estados dentro de los contratos.

# **Estructuras de control**



## If / else

Es la estructura condicional más utilizada.

Permite ejecutar una acción si la condición de entrada se cumple, y nos permite ejecutar acciones alternativas en caso de que esta condición no se cumpla.



# For

Es una estructura de bucle en la cual ejecutaremos una acción repetidas veces hasta que la condición de corte se cumpla.

Siempre se sabe la cantidad de ejecuciones en el momento previo a su comienzo.



# While

Es una estructura de bucle en la cual ejecutaremos una acción hasta que la condición de corte que se especifica al **comienzo** se cumpla.

No siempre se sabe la cantidad de ejecuciones en el momento previo a su comienzo y puede ejecutarse 0 veces.

# Do...While

Es una estructura de bucle en la cual ejecutaremos una acción hasta que la condición de corte que se especifica al **final** se cumpla.

No siempre se sabe la cantidad de ejecuciones en el momento previo a su comienzo y se ejecuta al menos una vez.



# Eventos



# Eventos

Un evento es una acción que ocurre en el contrato la cual queremos que sea notificada fuera del mismo.



# Eventos


Los eventos no tienen una función en especial dentro de los contratos.

Su objetivo es que sean integrados a componentes externos para poder ejecutar acciones externas como enviar una alerta a un usuario.



# Eventos

Debemos considerar que las llamadas a los eventos van a consumir gas y por lo tanto tienen un costo asociado. Se recomienda utilizarlos solo cuando sean necesarios.



# Funciones en Solidity

Implementando el  
comportamiento de un contrato



# Funciones



# Funciones

Son piezas de código que se identifican con un nombre y que pueden ser invocadas desde otras partes del contrato o fuera del mismo para realizar una acción.

**El ingreso de parámetros es opcional al igual que el retorno de un valor.** Es decir, puede que una función no retorne valores y solo ejecute una acción.



# Tipos de función

- **public**: son accesibles desde todo ámbito posible.
- **private**: solo son accesibles desde el mismo contrato.
- **internal**: solo son accesibles desde el mismo contrato y sus contratos derivados.
- **external**: solo accesibles desde fuera del contrato.







# Tipos de función

- **view**: indica que la función es de sólo lectura y no modifica el estado del contrato.
- **pure**: indica que no se accede a ningún valor del estado del contrato (o sea sus variables).



# Modificadores



# Modificadores

Permiten ejecutar código al inicio de una función y en general su uso está ligado a la restricción de permisos para la llamada de las mismas.

# **Manejo de errores**



# Manejo de errores

- **Assert:** permite evaluar un valor. Es utilizado principalmente para las pruebas de contratos.
- **Revert:** interrumpe la ejecución de una función y revierte todos los cambios realizados hasta el momento.
- **Require:** evalúa una condición y en caso de no cumplirla ejecuta un **revert**. Puede mostrar un mensaje.



# Gestionando costos

Gas, comisiones y transferencias





# **Tipos de almacenamiento**

# Tipos de almacenamiento

- **Storage:** es el almacenamiento persistente de los contratos, por lo tanto su costo es superior.
- **Memory:** almacenamiento temporal, su contenido está ligado al ámbito de la función que lo declara. Su costo es menor al de **storage**.
- **Calldata:** es donde se alojan los parámetros y su comportamiento es similar al de **memory**.

# **Gas y comisiones**



# Gas

El gas nos indica **cuánto procesamiento es necesario para ejecutar una acción.**

Este indicador es siempre el mismo ante la misma acción.



# Gas price

**Es el precio actual que tiene el gas según el uso de la red.**

Este valor puede variar en el paso del tiempo.



# Gas fee

Es la **comisión que se le paga al minero por validar nuestra transacción.**

Cuando se implementa un contrato o se escribe un valor también estamos realizando una transacción.



# Gas total

Es la cantidad total que se debe pagar por la operación que realizamos. El cálculo es:

$$\text{Gas total} = (\text{Gas} \times \text{Gas price}) + \text{Gas fee}$$





# **Transferencias de Ether desde un contrato**

# Transferencia de Ether desde un contrato

- **Send:** envía un monto de Ether a una dirección y retorna “false” si no se puede realizar la acción.
- **Transfer:** envía un monto de Ether a una dirección e interrumpe la ejecución de la función si no se puede realizar la acción.
- **Call:** envía un monto de Ether a una dirección y retorna el resultado de la operación.

# Diferencias

- **Send** y **transfer** requieren que las direcciones sean de tipo **payable**.  
**Call** no lo requiere.
- **Send** y **transfer** tienen un límite de gas fijo de 2300 de gas.  
**Call** no tiene límite de gas por defecto, pero se puede configurar.
- **Call** permite llamar a funciones si la dirección especificada es un contrato.

# **Recibir Ether desde un contrato**

# ■ Formas de recibir Ether desde un contrato

- **Receive**: función opcional que se ejecuta cuando se recibe una transferencia de Ether sin parámetros.
- **Fallback**: función opcional que se ejecuta cuando se recibe una transferencia de Ether con parámetros.
- **Función payable**: se puede recibir Ether en una función si se le especifica el tipo **payable**.



# ■ Formas de recibir Ether desde un contrato

- Cuando se envía una transferencia lo primero que se intenta es verificar si existe una función con la firma especificada.
- Si no se encuentra, se buscará una función **fallback** que reciba parámetros.

# ■ Formas de recibir Ether desde un contrato

- Si no tiene parámetros, se buscará una función de **receive** y en caso de no existir se buscará una función **fallback** sin parámetros.





# Conceptos avanzados

Librerías, herencia y tokens



# **Manejo de dependencias y librerías**

# Importar una dependencia

- Con la sentencia “***import***” podemos hacer referencia a un contrato que esté definido en el mismo ámbito en el que estemos trabajando.
- También podemos importar contratos que se encuentren en un repositorio o en un paquete como **npm**.

# Importar una dependencia

- Además de contratos podemos importar librerías que son similares a los contratos, pero no contienen estado y solo brindan utilidad.

# Herencia



# Herencia

- El concepto es similar al de la programación orientada a objetos y la identificaremos con la sentencia “*is*”.
- Si un contrato tienen un constructor con parámetros debemos indicar **qué valores debe tomar ese constructor para poder derivarse.**



# Override

- Podemos definir funciones de tipo “***virtual***” para indicar que pueden ser redefinidas en sus contratos derivados.
- Cuando se redefine una función se debe indicar el tipo “***override***” en su declaración.



# Override

- Si una función virtual no define implementación, el contrato se convierte en un **contrato abstracto**.





# Interfaces

- Nos permiten definir el comportamiento que queremos que tenga un contrato.
- Se implementa igual que la herencia de contratos.
- Sus funciones no tienen implementación, solo encabezados.



# Super

Podemos hacer referencia a algún elemento o función de la clase superior por medio de la sentencia “***super***”.

# Polimorfismo

# Instanciar un contrato en otro contrato

- Dentro de un contrato podemos hacer referencia a otro contrato ya implementado en la red a través de su dirección.
- Para esto podemos utilizar el tipo de contrato al que referenciamos o bien alguna de sus clases superiores.



# Polimorfismo

- Capacidad de poder utilizar contratos derivados en estructuras superiores.

# Tokens



# Tokens

- Son una representación de un elemento que tiene un valor en un contexto determinado.
- En general esta representación tiene menos valor que el objeto original.
- Existen diferentes clasificaciones para los tokens.



# Clasificación

- **Fungibles:** son elementos reemplazables por otros con las mismas características.  
Además son divisibles y podemos consumir solo una fracción.
- **No fungibles:** son elementos que pueden variar de valor respecto de elementos de iguales características. No son divisibles.





# ERC-20

- Es el estándar que representa a los tokens fungibles.
- Al ser un estándar no cuenta con una implementación, sino que solo define una interfase.
- Existen otros estándares que implementan tokens fungibles, pero todos ellos mantienen la compatibilidad contra el token ERC-20.



# ERC-721

- Es el estándar que representa a los tokens no fungibles.
- Al igual que el ERC-20 solo define una interfase.
- A diferencia de los tokens fungibles, estos tienen un identificador único conocido como tokenId.

The logo consists of three small squares stacked vertically: a light green one at the top, a yellow one in the middle, and a dark green one at the bottom. 

# TokenUri

- Contiene información no técnica acerca de un token.
- En general tiene que ver con información estética o visual relacionada.
- Fue introducida por el estándar ERC-721, pero se utiliza en otros tipos de token.

**ABI**



# ABI

- **ABI** viene de Application Binary Interface.
- Es una interfase que nos detalla qué **definiciones tiene el contrato y sus funciones**, sin conocer su implementación.
- Nos permite saber qué forma tiene un contrato para poder interactuar con sus funciones, especialmente si estamos construyendo la capa de usuario.



Nos permite saber qué forma tiene un contrato para poder interactuar con sus funciones, especialmente si estamos construyendo la capa de usuario.



# Despedida

Próximos pasos a seguir en la carrera  
del desarrollador de blockchain

