CHEATSHEET SOLIDITY

Smart Contracts y Blockchain con Solidity de la A a la Z

SOLIDITY

Solidity es un lenguaje de alto nivel orientado a contratos. Su sintaxis es similar a la de JavaScript y está enfocado específicamente a la Máquina Virtual de Etehreum (EVM). Esta hoja de trucos presenta las bases para poder programar en Solidity

ANTES DE EMPEZAR

pragma solidity <rango versiones>;	Establecer un rango de versiones que debe usar el compilador.
pragma solidity <version>;	Establecer la version que debe usar el compilador
import "./<nombre archivo>.sol";	Importar un archivo en Solidity
import <subset> from "./<nombre archivo>.sol";	Importar un subconjunto de contratos o librerias del archivo especificado
pragma experimental ABIEncoderV2;	Necesario para poder usar la funcion abi.encodePacked()

BASES

contract <nombre contrato> {...}	Crear un contrato
constructor(<argumentos>*) public {...}	Declarar un constructor del contrato
//, /*...*/	Comentarios de una línea y de bloque respectivamente

FUNCIONES DISPONIBLES GLOBALMENTE

block.blockhash(blockNumber)	Devuelve el hash de un bloque dado
block.coinbase	Devuelve la dirección del minero que está procesando el bloque actual
block.difficulty	Devuelve la dificultad del bloque actual
block.gaslimit	Devuelve el límite de gas del bloque actual
block.number	Devuelve el número del bloque actual
block.timestamp	Devuelve el timestamp del bloque actual en segundos
msg.data	Datos enviados en la transacción
msg.gas	Devuelve el gas que queda
msg.sender	Devuelve el remitente de la llamada actual
msg.sig	Devuelve los cuatro primeros bytes de los datos enviados en la transacción
msg.value	Devuelve el numero de Wei enviado con la llamada
now	Devuelve el timestamp del bloque actual
tx.gasprice	Devuelve el precio del gas de la transacción
tx.origin	Devuelve el emisor original de la transacción
keccak256(abi.encodePacked(...))	Cómputo del hash SHA256

CHEATSHEET SOLIDITY

Smart Contracts y Blockchain con Solidity de la A a la Z

VARIABLES	
<tipo dato> <nombre variable>;	Declarar una variable
<tipo dato> <nombre variable> = <valor>;	Declarar y inicializar una variable

TIPOS DE DATOS	
uint<x>	Enteros positivos de x bits (x varia de 8 a 256 en múltiplos de 8)
int<x>	Enteros con signo de x bits (x varia de 8 a 256 en múltiplos de 8)
bool	Tipo de dato de lógica binaria (true o false)
string	Cadena de texto
bytes<x>	Tipo de dato de más bajo nivel, bytes (x varia de 1 a 32 de uno en uno)
address	Tipo de dato para las direcciones en Ethereum.
<x> [seconds, minutes, days, weeks, years]	Unidades de tiempo en Solidity. Devuelven un entero con el número de segundos

CASTEO DE VARIABLES	
uint<x>(<dato uint<y>>)	Transformar una variable entera positiva con y bits a una variable entera positiva con x bits
int<x>(<dato int<y>>)	Transformar una variable entera con signo con y bits a una variable entera con signo con x bits
int<x>(<dato uint<y>>)	Transformar una variable entera positiva con y bits a una variable entera con signo con x bits
uint<x>(<dato int<y>>)	Transformar una variable entera con signo con y bits a una variable entera positiva con x bits

MODIFICADORES DE VARIABLES	
public	Creación de una función getter. Visibilidad total
private	Visibilidad solo desde dentro del contrato
internal	Accesibilidad interna (desde dentro del contrato y contratos derivados).
payable	Solo disponible para address. Permite enviar y recibir ether
memory	Guardado temporal
storage	Guardado permanente en la Blockchain

OPERADORES	
+, -, * , / ,%, **	Suma, resta, multiplicación, división, módulo, exponenciación
!, &&, ||, ==, !=	Negación, and, or, igualdad, inigualdad
>, >=, <, <=, ==, !=	Mayor estricto, mayor o igual, menor estricto, menor o igual, igualdad, inigualdad

CHEATSHEET SOLIDITY

Smart Contracts y Blockchain con Solidity de la A a la Z

ESTRUCTURAS

struct <nombre estructura>{
<tipo dato 1> <nombre>;
<tipo dato 2> <nombre>;
...
}
	Declarar un tipo de dato complejo con sus propiedades

<nombre estructura>(<propiedades>);	Definir un nuevo dato <nombre estructura>
<nombre>.<propiedad>;	Acceder a una propiedad de un tipo de dato estructura

MAPPINGS

mapping (<key> => <value>) <nombre>;	Declarar un mapping
<nombre>[_key] = _value;	Guardar un dato en el mapping
<nombre>[_key];	Acceder al valor asociado a una clave

ARRAYS

<tipo dato> [<longitud>] <nombre>;	Declarar un array de longitud fija
<tipo dato> [ ] <nombre>;	Declarar un array dinámico
<tipo dato> [<longitud>*] <nombre> = [<valores>];	Declarar y inicializar un array fijo o dinámico
<nombre>[<índice>];	Acceder al valor de una posición del array
<nombre>[<índice>] = <valor>;	Guardar un valor en una posición del array
<nombre>.push(<dato>);	Añadir un dato al final del array con la función push() [Solo disponible para arrays dinámicos]
<nombre>.length;	Devuelve la longitud del array

CHEATSHEET SOLIDITY

Smart Contracts y Blockchain con Solidity de la A a la Z

FUNCIONES	

function <nombre funcion>(<tipos parámetros>) [public | private | internal | external] [view | pure | payable]* [returns (<return types>)]*{
...
return (<valores retorno>)*;
}
	Declarar una función

<tipos parámetros>	Tipos de datos que se proporcionan a la función en su llamada
public	Modificador de visibilidad. Forman parte de la interfaz del contrato y son accesibles fuera y dentro del contrato
private	Modificador de visibilidad. Solo son accessibles dentro del contrato
internal	Modificador de visibilidad. Solo son accessibles internamente (dentro del contrato y contratos derivados)
external	Modificador de visibilidad. Parecido a public. Solo son accesibles fuera del contrato
view	No se modifica los datos, solo se acceden a ellos
pure	No se acceden a los datos. La funcion depende de los parámetros
payable	Permite recibir ether
returns(<return types>)	Declarar los tipos de datos que devolverá la función
return <valores retorno>;	Devuelve los valores declarados en la definición de la función
require(<condicion>, ["Mensaje"]*);	Comprobar una condición para seguir con la ejecución de la función

MODIFIER	

modifier <nombre modificador> (<parámetros>) *{
require(<condicion>, ["Mensaje"]*);
_;
}
	Declarar un modifier

function <nombre funcion> ... [<nombre modificador>(<parámetros>)*]*	Usar un modifier en una función

BUCLES	
if(<condición>){...}else{}	Declarar un if
for(<iniciar contador>; <comprobar contador>; <aumentar contador>){...}	Declarar un bucle for
while(<condición>){...}	Declarar un bucle while
break;	Detener la ejecución de un bucle y salir de él

EVENTOS	
event <nombre evento> (<types>);	Declarar un evento
emit <nombre evento> (<valores>);	Emitir un evento

CHEATSHEET SOLIDITY

Smart Contracts y Blockchain con Solidity de la A a la Z

HERENCIA

contract <nombre contrato> is <nombre contrato padre>{...}	Declarar un contrato hijo que hereda las funciones y variables pertinentes del contrato padre

LIBRERÍAS

library <nombre librería>{...}	Declarar una librería
using <nombre librería> for <tipo dato>	Usar una librería en un contrato

INTERFAZ

contract <nombre interfaz>{
function <nombre función> ... ;
}
	Declarar una interfaz con las funciones que usaremos del contrato con el que queremos interactuar

<nombre interfaz> <nombre puntero> = <nombre interfaz>(dirección contrato);	Declarar un puntero que apunta al contrato con el que queremos interactuar
<nombre puntero>.<función>(<parámetros>);	Usar las funciones definidas en la interfaz

FACTORY

function <nombre Factory>() public {
address <dirección nuevo contrato> = address (new <nombre contrato>(<parametros>));
}
	Declarar una función factory

contract <nombre contrato> {
constructor(<parámetros>) public {...}
}
	Contrato a partir de la función factory