Assignment 09

1) What are the different ways in which import can be used in Solidity?

Ans)

Global Imports:

The statement below will import all Solidity objects found in

import “./MySolidityFile.sol”;

When the Solidity compiler encounters this statement, two things happen.

All global symbols (what I call “Solidity Objects”) defined in “./MySolidityFile.sol” get imported into the current global scope.

All global symbols imported inside “./MySolidityFile.sol” get also imported into the current global scope.

This is very different from ES6! In ES6, when you import a module, you import what is defined inside this module, not everything else imported within this module

Specific Imports:

Specific imports are closer to the ES6 syntax. You can be more specific and import what you need from a Solidity file.

Import { Something } from “./MySolidityFile.sol”;

You can then mention within the curly braces { } the specific symbols/objects that you want to import and use. For instance, if the file specified in the import path contains multiple contract, you can determine which contracts you wish to import exactly with this syntax.

Let’s look at the following example from Solmate, a smart contract library under development that can be used to build modern and gas optimised contracts.

2) How to use Openzeppelin's SafeMath library?

Ans: Wrappers over Solidity’s arithmetic operations with added overflow checks.

Arithmetic operations in Solidity wrap on overflow. This can easily result in bugs, because programmers usually assume that an overflow raises an error, which is the standard behavior in high level programming languages. SafeMath restores this intuition by reverting the transaction when an operation overflows.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/SafeMath.sol

3) Can we use multiple SPDX licenses? Yes/No, Give reason. (Try out in Remix)

Ans: No, Only one license identifier is used. Using multiple contract file the compiler will throw an error.

4) Develop a Smart Contract called MyFriends OR PGDFBD_Batchmates

Ans:

```
//SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.7;
contract PGDFBD_Batchmate{
    uint public StudentNo;
    string public StudentName;

    function set(uint a) public{
        StudentNo=a;
    }

    function get() public view returns (uint) {
        return StudentNo;
    }

    function setinfo(string memory name) public {
        StudentName=name;
    }

    function getinfo() public view returns (string memory){
        return StudentName;
    }
}
```

5) What is the largest number that can be represented using 256 bits?

Ans: The maximum value of an unsigned 256-bit integer is 2256 − 1,

*written in decimal as 115,792,089,237,316,195,423,570,985,008,687,907,853, 269,984,665, 640,564,039,457,584,007,913,129,639,935

* approximately as 1.1579 x 1077.

6) What are contract types & integer literals in Solidity?

Ans: Integer literals are formed from a sequence of digits in the range 0-9. They are interpreted as decimals. For example, 69 means sixty nine. Octal literals do not exist in Solidity and leading zeros are invalid.

Integer literals are formed from a sequence of digits in the range 0-9. They are interpreted as decimals. For example, 69 means sixty nine. Octal literals do not exist in Solidity and leading zeros are invalid.

Scientific notation in the form of 2e10 is also supported, where the mantissa can be fractional but the exponent has to be an integer. The literal MeE is equivalent to M * 10**E. Examples include 2e10, -2e10, 2e-10, 2.5e1.

Underscores can be used to separate the digits of a numeric literal to aid readability. For example, decimal 123_000, hexadecimal 0x2eff_abde, scientific decimal notation 1_2e345_678 are all valid. Underscores are only allowed between two digits and only one consecutive underscore is allowed. There is no additional semantic meaning added to a number literal containing underscores, the underscores are ignored.

Number literal expressions retain arbitrary precision until they are converted to a non-literal type (i.e. by using them together with anything other than a number literal expression (like boolean literals) or by explicit conversion). This means that computations do not overflow and divisions do not truncate in number literal expressions.

For example, (2**800 + 1) - 2**800 results in the constant 1 (of type uint8) although intermediate results would not even fit the machine word size.

Furthermore, .5 * 8 results in the integer 4 (although non-integers were used in between).

6) Perform conversion from address to address payable.address to uint160,address to bytes20.

The address type comes in two flavours, which are largely identical:

address: Holds a 20 byte value (size of an Ethereum address).

address payable: Same as address, but with the additional members transfer and send. The idea behind this distinction is that address payable is an address you can send Ether to, while you are not supposed to send Ether to a plain address, for example

because it might be a smart contract that was not built to accept Ether.

Type conversions:

Implicit conversions from address payable to address are allowed, whereas conversions from address to address payable must be explicit via payable(<address>). Explicit conversions to and from address are allowed for uint160, integer literals, bytes20 and contract types. Only expressions of type address and contract-type can be converted to the type address payable via the explicit conversion payable. For contract-type, this conversion is only allowed if the contract can receive Ether, i.e., the contract either has a receive or a payable fallback function. Note that payable(0) is valid and is an exception to this rule.

As described, hex literals of the correct size that pass the checksum test are of address type. No other literals can be implicitly converted to the address type.

Explicit conversions to address are allowed only from bytes 20 and uint160.

bytes20(address) is capturing top (left) 20 bytes, while: uint160(uint256(address)) Is capturing lowest (right) 20 bytes.

An address a can be converted explicitly to address payable via payable(a).

7) What is the difference between transfer & send when it comes to addresses?

Transfer: the receiving smart contract should have a fallback function defined or else the transfer call will throw an error. There is a gas limit of 2300 gas, which is enough to complete the transfer operation. It is hardcoded to prevent reentrancy attacks.

Send: It works in a similar way as to transfer call and has a gas limit of 2300 gas as well. It returns the status as a boolean.

8) Difference between call, staticcall & delegatecall

Call: The call method returns a pair of values: a boolean informing whether the function was executed successfully or not, and a value of type bytes with the function's return, also ABI encoded.

Staticall: Staticcall is a method similar to call, but it does not allow changing the state

of the blockchain. This means that we cannot use staticcall if the called function changes some state variable, for example.

In the figure below, we replaced the method call with staticcall. By doing this, the compiler gives us a warning that the function callSetNumber can be declared as view, as staticcall does not allow changing the state of the blockchain.

If we use staticcall to invoke a function that changes the state of the blockchain, the function invocation will not be successful.

Delegatecall: It is also possible to execute a function in another contract, but in such a way that it changes the state variables of the calling contract. For this purpose, the method delegatecall is used.

9) Check does your address passes the Checksum test.

Yes, it is. Go to etherscan.io > paste your address > Overview > the ethereum checksum address will appear at the top.

10) What is address checksum test?

A checksum is a string of numbers and letters that's used to "check" whether data or a file has been altered during storage or transmission. Checksums often accompany software downloaded from the web so that users can ensure the file or files were not modified in transit. If the checksum from the software vendor matches the checksum of the downloaded installation files on your computer, then no errors or modifications were made. If the checksums don't match up, the download might have been corrupted or compromised by hackers.

11) Deploy Enums Test Contract on Goerli Testnet.

ANS.

0x74e3ac3522c56cdee180788ffa37bef943ccd861ffbcb5411339ed57907ffa21