LAB 10-07-2023

Primitive Data Types

In Solidity, there are different types of variables that you can use to store and manipulate data. These types can be categorized into value types, reference types, mapping types, and function types. In this lab, we will focus on the value types:

1 Boolean:

- The bool type represents a binary variable that can hold either **true or false values.**
- You can use logical operators like ! (logical NOT), && (logical AND), and || (logical OR) to perform boolean operations.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract BooleanOperators {

    bool public _bool = true;

    function logicalNot() public view returns (bool) {

        return !_bool;                                }

    function logicalAnd() public view returns (bool) {

        return _bool && !_bool;                      }

    function logicalOr() public view returns (bool) {

        return _bool || !_bool;                      }

    function equality() public view returns (bool) {

        return _bool == !_bool;                      }

    function inequality() public view returns (bool) {

        return _bool != !_bool;                      }

}
```

TASK Make a contract named ANDGate and confirm the output of AND gate as following

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

2 Integers:

Solidity provides both signed and unsigned integer types.

- The int type is used for signed integers, while the uint type is used for unsigned integers.
- You can specify the size of integers using suffixes like int8, uint256, etc.
- You can perform arithmetic operations and comparison operations on integer types.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract IntegerVariations {

    int8 public int8Var;

    int16 public int16Var;

    int32 public int32Var;

    int64 public int64Var;

    int128 public int128Var;

    int256 public int256Var;


    uint8 public uint8Var;

    uint16 public uint16Var;

    uint32 public uint32Var;

    uint64 public uint64Var;

    uint128 public uint128Var;

    uint256 public uint256Var;

}
```

TASK Explore the variations of integer data types in Solidity and write a contract that finds the minimum and maximum values range using the

type(datatype).min and type(datatype).max properties.

Example:

```
// minimum and maximum of int

int public minInt8 = type(int8).min;

int public maxInt8 = type(int8).max;
```

3 Addresses:

- The address type is used to store Ethereum addresses.
- It represents a **20-byte value** and can be used to interact with other contracts or send and receive Ether.
- Solidity also provides a variant called address payable that includes additional functions for transferring Ether.

```
// Address

  address public address = 0x7A58c0Be72BE218B41C608b7Fe7C5bB630736C71;

  // Members of address

  uint256 public balance = address1.balance; // balance of address
```

TASK Declare an Ethereum address variable called **myAddress** and assign it with your Metamask Wallet's public address.

4 Bytes

Solidity presents two type of bytes types :

- fixed-sized byte arrays

- dynamically-sized byte arrays.

4.1 Fixed-Size Byte Arrays:

Solidity allows you to define fixed-size byte arrays using types like bytes1, bytes32, etc.

These arrays can hold a specific number of bytes and are useful for working with **binary data**.

```
// Fixed-size byte arrays
    bytes32 public byte32 = "MiniSolidity";
    bytes1 public b1 = byte32[0];
```

4.2 Dynamically-Sized Byte Array

Bytes and strings are special types used for handling arbitrary-length raw byte data.

The bytes type represents a dynamic array of bytes, and it is essentially a shorthand for byte[].

It can have a variable length, including a length of zero, and supports operations like appending a byte to the end of the array.

It's important to note that bytes is not considered a value type in Solidity.

TASK Observe and deploy the following contract.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;
contract darray {
    uint256[] myarray;
    function addtoarray(uint256 number) public returns (uint256, uint256)
{
        myarray.push(number);
        return (number, myarray.length);            }
```

```
    function getarrayvalue(uint256 index) public view returns (uint256) {
        return myarray[index];                          }
    function popValue() public  {
         myarray.pop();                                 }
    function deleteItem(uint256 index) public  {
         delete myarray[index];                         }          }
```

Note: Unassigned variables have default values.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;
contract DefaultValues {
    bool public defaultBoo; // false
    uint public defaultUint; // 0
    int public defaultInt; // 0
    address public defaultAddr; //0x0000000000000000000000000000000000000000
}
```

TASK Deploy the above contract and observe the default values.

ByteCode

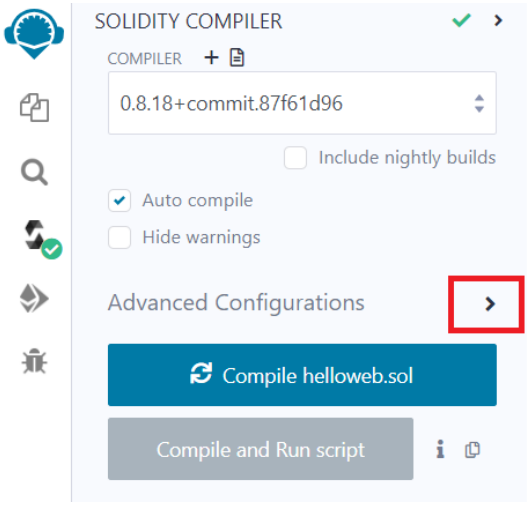
Solidity bytecode is the **low-level machine code** representation of a smart contract, generated by the Solidity compiler.

Bytecode is executed by the **Ethereum Virtual Machine (EVM)** and deployed on the blockchain, serving as the immutable executable code for the contract's logic.

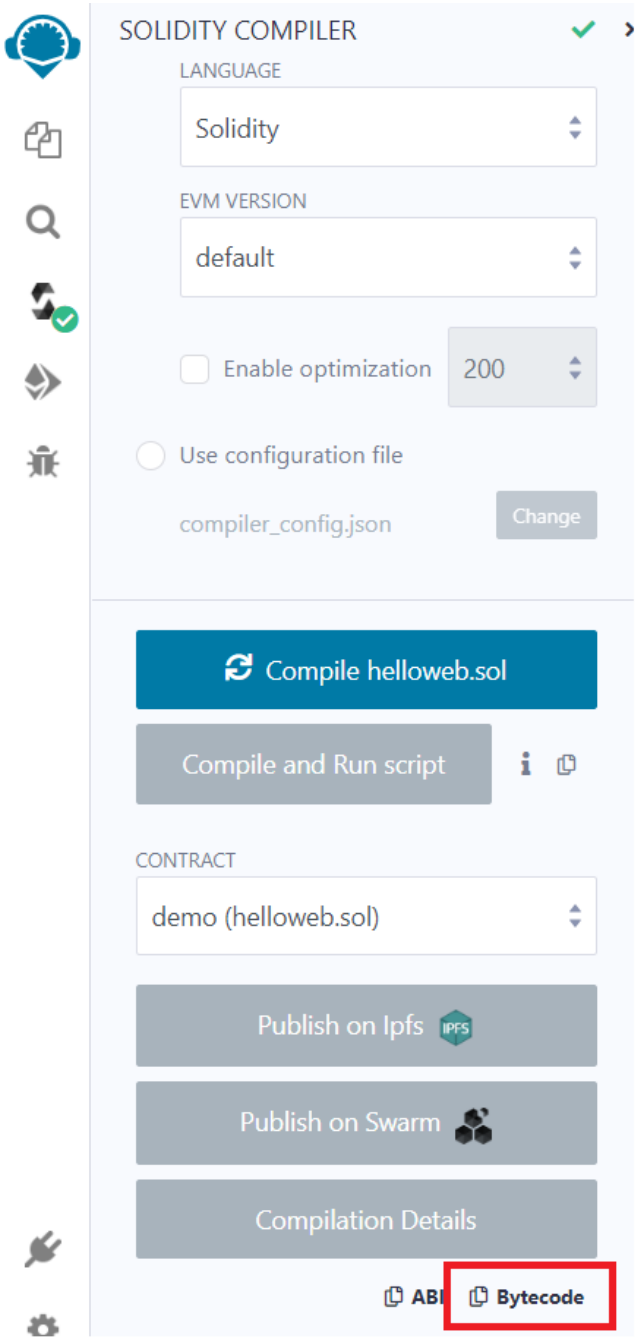
Task Observe the bytecode of the following code

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract demo {

    string public greet = "Hello Web 3.0";     }
```

Step 01: Make sure you have enabled **Auto Compile.** Click on Arrow button next to **Advanced Configurations** as highlighted

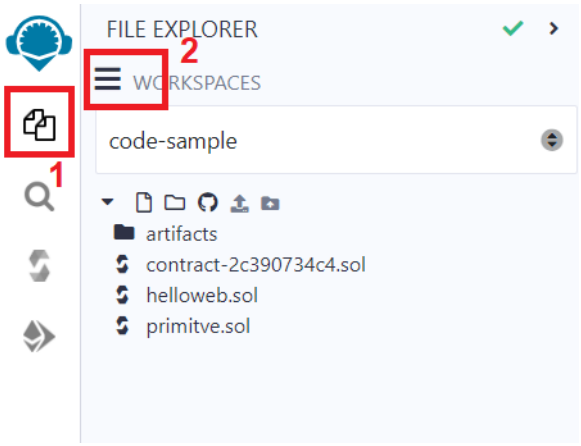


Step 02 Scroll down, Copy the bytecode by clicking on **Bytecode** as highlighted. Paste it on some **.docx** file and observe it.

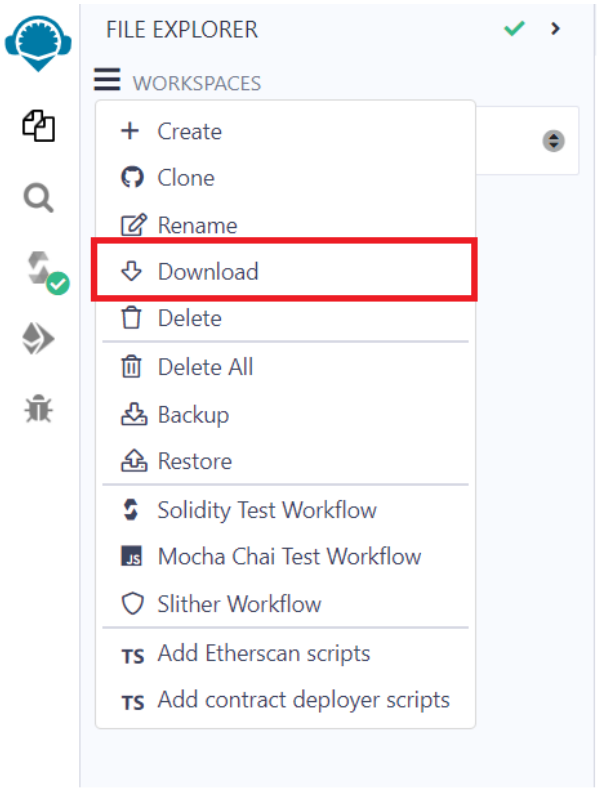

SOLIDITY COMPILER
LANGUAGE
Solidity
EVM VERSION
default
Enable optimization
200
Use configuration file
compiler_config.json
Change
Compile helloweb.sol
Compile and Run script
CONTRACT
demo (helloweb.sol)
Publish on Ipfs
Publish on Swarm
Compilation Details
ABI
Bytecode

Downloading the Remix Project

Step 01: Navigate to **Workspaces** in the **File Explorer.**



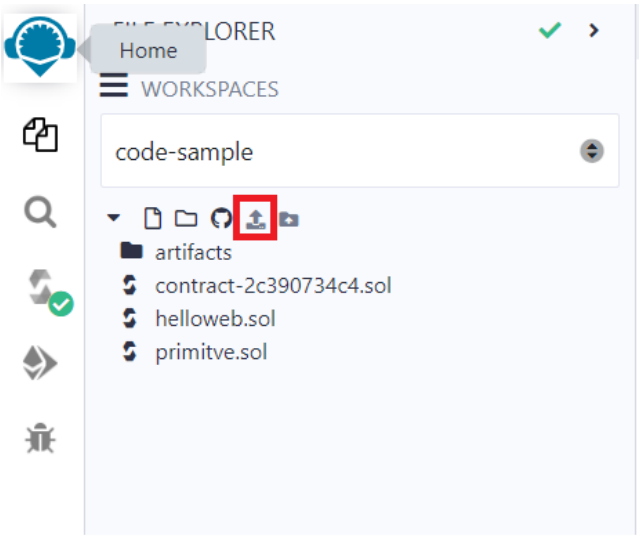
Step 02: As soon as you press the **Workspace Actions,** a drop down menu will be appear. Select **Download** from that menu.



Step 03: A **zip** file will be downloaded on your local disk. Extract it.

Importing the Downloaded Project:

You can easily import your project, available on your local disk , by clicking on **Upload Files** button in File Explorer as highlighted.



Please attempt the corresponding quiz related to this hands on exercise by accessing the link below:
https://forms.gle/QgwmS3EB849HnEgP9

For complete documentation, visit https://remix-ide.readthedocs.io/en/latest/