Ethereum Virtual Machine

Solidity Programming Toolset

Solidity: Part I

01 Interactive Development Environment

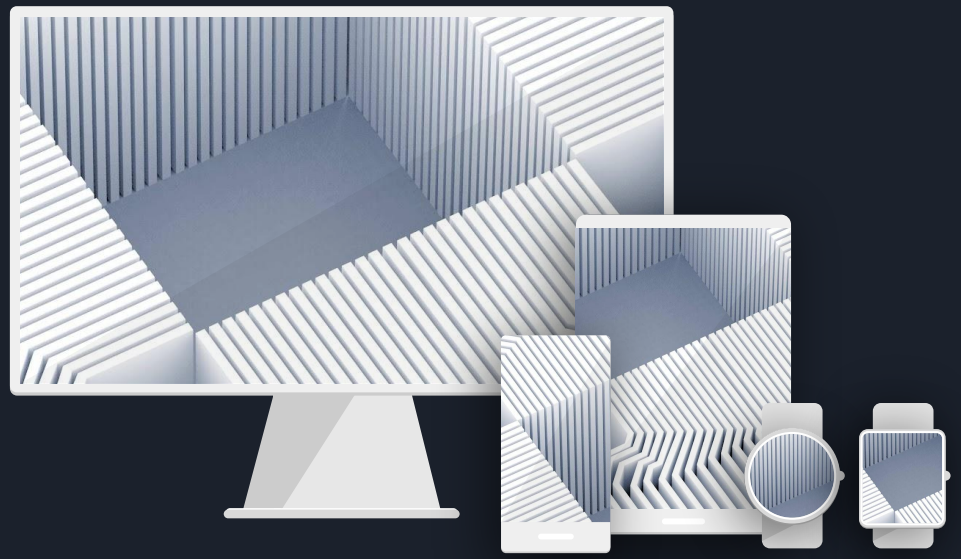
02 Types and Variables

03 Operators, Loops, and Decision Making.

https://remix.ethereum.org/

IDE

Online, Windows, Mac, Linux.



Solidity Review

Basic Types/ Variables/ Operations

02: Types

- bool
- int/uint
- int8 to int256
- uint8 to uint256
- fixed/unfixed
- fixedMxN
- ufixedMxN

02: Variables

- State Variables
- Local Variables
- Global Variables

State Variables

Public – Public state variables can be accessed internally and externally.

– Similar to the "public static" state

Internal – Internal state variables can be accessed from the current contract.

– also are accessible by a related contract.

Private – Private state variables can be accessed from the current contract.

– are NOT accessible by a related contract.

STATE VARIABLE TYPES

public / internal / private

```
pragma solidity ^0.5.0;
contract C {
   uint public data = 30;                          // public
   uint internal iData= 10;                        // internal

   function x() public returns (uint) {
      data = 3;                                    //
      return data;
   }
}
contract Caller {
   C c = new C();
   function f() public view returns (uint) {
      return c.data();
   }
}
contract D is C {
   function y() public returns (uint) {
      iData = 3;                                   //
      return iData;                                //
   }

   function getResult() public view returns(uint){
      uint a = 1;                                  //
      uint b = 2;                                  //
      uint result = a + b;                         //
      return iData;                                //
   }
}
```

02: State Variable

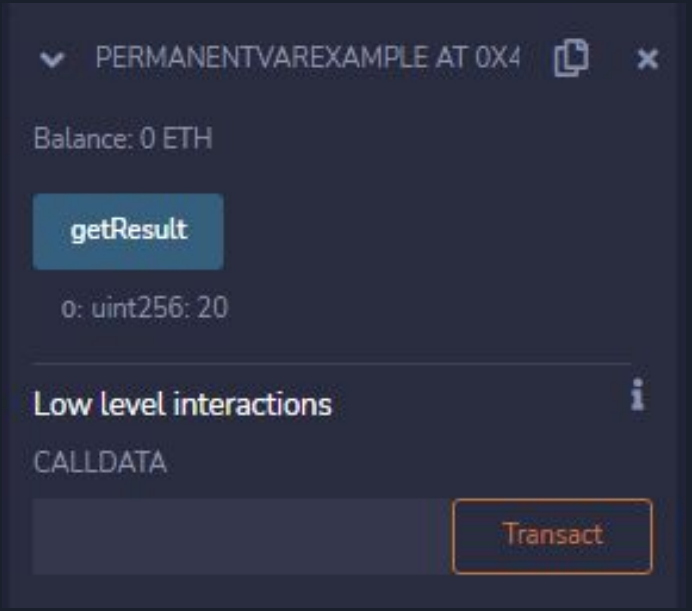
```
pragma solidity ^0.5.0;                 // pragma version of solidity.

contract PermanentVarExample{
   uint storedData;                     // State variable declared.

   constructor() public {
      storedData = 10;                  // State variable assigned a value of 10.
   }
}
```

02: Local Variable

```
pragma solidity ^0.5.0;                                    // solidity version

contract PermanentVarExample {
   uint storedData;
                                                           // state variable declared.
   constructor() public {
      storedData = 10;
   }

   function getResult() public view returns(uint){
      uint a = 10;                                         // local variables
      uint b = 10;
      uint result = a + b;
      return result;                                       // access this from outside function.
   }

}
```

02: Local Variable

```
pragma solidity ^0.5.0;

contract PermanentVarExample{
   uint storedData;


   constructor() public {
      storedData = 10;
   }

   function getResult() public view returns(uint){
         uint a = 10;
         uint b = 10;

         uint result = a + b;
         return result;
   }

}
```



02: Reserved Keywords

abstract	after	alias	apply
auto	case	catch	copyof
default	define	final	immutable
implements	in	inline	let
macro	match	mutable	null
of	override	partial	promise
reference	relocatable	sealed	sizeof
static	supports	switch	try
typedef	typeof	unchecked	

Mathematical Operators

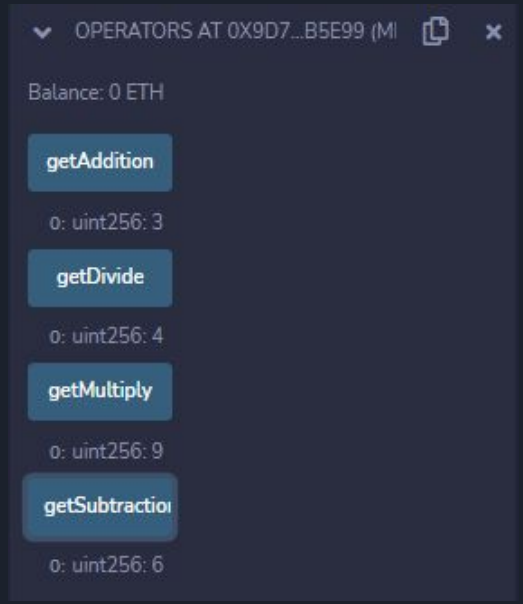
03: Operators

- ➔ Arithmetic
- ➔ Logical
- ➔ Conditional
- ➔ Bitwise

03.01 Arithmetic

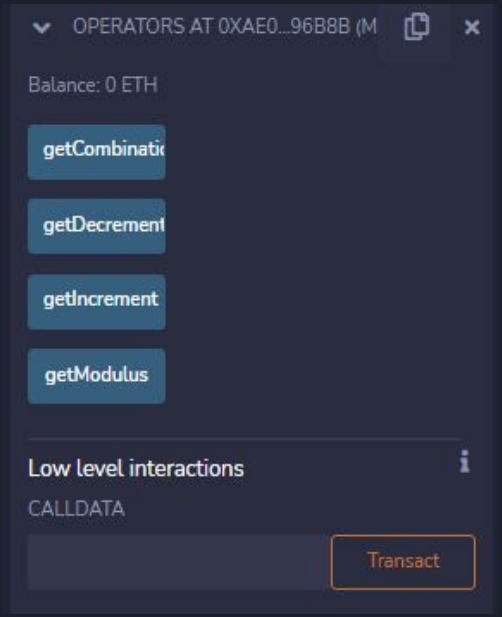
```
pragma solidity ^0.5.0;

contract Operators {
   uint storedData;
   constructor() public {
      storedLcl = 10;
   }

   function getAddition() public view returns(uint){
      uint a = 1;
      uint b = 2;
      uint result = a + b;
      return result;
   }

   function getSubtraction() public view returns(uint){
      uint a = 1;
      uint b = 7;
      uint result = b - a;
      return result;
   }

   function getMultiply() public view returns(uint){
      uint a = 9;

     uint b = 1;
      uint result = a*b;
      return result;
   }

   function getDivide() public view returns(uint){
      uint a = 8;
      uint b = 2;
      uint result = a/b;
      return result;
   }
}
```

03.01 Arithmetic




```
pragma solidity ^0.5.0;

contract Operators {
    uint storedData;
    constructor() public {
        storedLcl = 10;
    }

    function getAddition() public view returns(uint){
        uint a = 1;
        uint b = 2;
        uint result = a + b;
        return result;
    }

    function getSubtraction() public view returns(uint){
        uint a = 1;
        uint b = 7;
        uint result = b - a;
        return result;
    }

    function getMultiply() public view returns(uint){
        uint a = 9;
        uint b = 1;
        uint result = a*b;
        return result;
    }

    function getDivide() public view returns(uint){
        uint a = 8;
        uint b = 2;
        uint result = a/b;
        return result;
    }
}
```

03.02

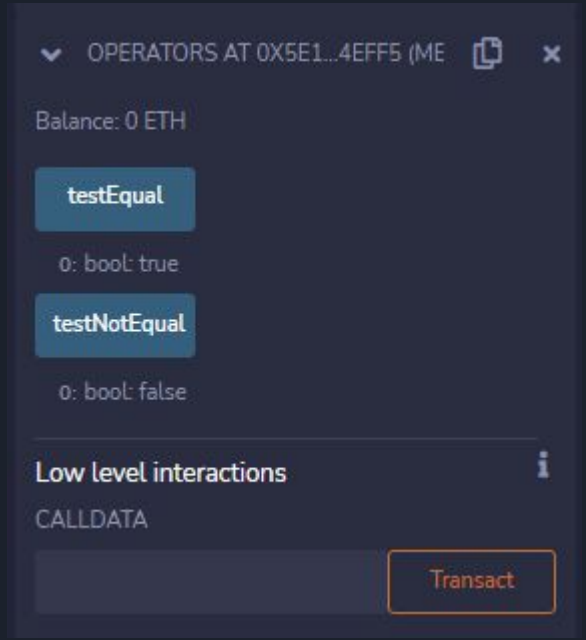



```
pragma solidity ^0.5.0;

contract Operators {
    uint storedData;
    constructor() public {
        uint storedLcl = 10;
    }

    function getModulus() public view returns(uint){
        uint a = 66;
        uint b = 46;
        uint result = a%b;
        return result;
    }

    function getIncrement() public view returns(uint){
        uint a = 21;
        return a++;
    }

    function getDecrement() public view returns(uint){
        uint r = 22;
        r--;
        return r;
    }

    function getCombination() public view returns(uint){
        uint a = 9;
        a++;
        uint b = 3;
        uint c = a%b;
        uint d = a+c;
        return d;
    }
}
```

03.03 Boolean Functions

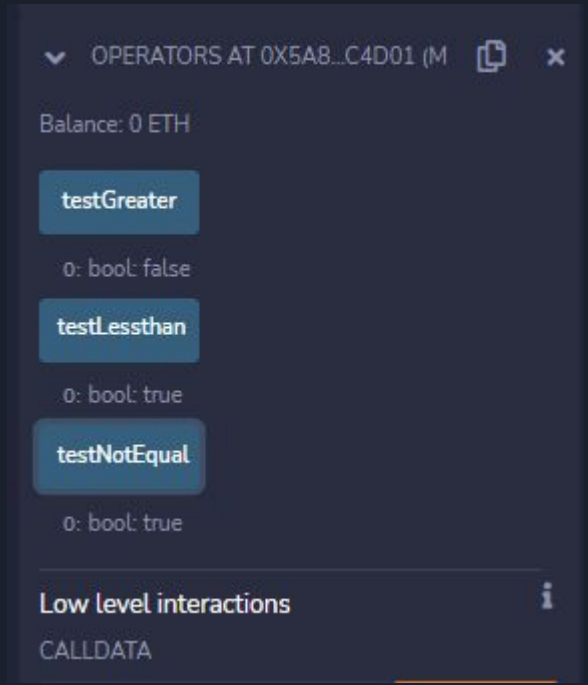
```
pragma solidity ^0.5.0;

contract Operators {

    constructor() public {
    }

    function testNotEqual() public view returns(bool){
        uint dataone = 20;
        uint datatwo = 21;
        if(dataone!=datatwo){
            return true;
            }
            else{
                return false;
            }
    }

    function testGreater() public view returns(bool){
        uint dataone = 20;
        uint datatwo = 20;
        if(dataone>=datatwo){
            return true;
            }
            else{
                return false;
            }
    }

    function testLessthan() public view returns(bool){
        uint dataone = 20;
        uint datatwo = 25;
        if(datatwo <=dataone ){
            return true;
            }
            else{
                return false;
            }
    }
}
```

03.03 Boolean Functions




```
pragma solidity ^0.5.0;

contract Operators {
    uint dataOne;
    uint dataTwo;
    constructor() public {
    }

    function testNotEqual() public view returns(bool){
        uint dataone = 20;
        uint datatwo = 21;
        if(dataone!=datatwo){
            return true;
            }
            else{
                return false;
            }
    }

    function testGreater() public view returns(bool){
        uint dataone = 20;
        uint datatwo = 20;
        if(dataone>datatwo){
            return true;
            }
            else{
                return false;
            }
    }

    function testLessthan() public view returns(bool){
        uint dataone = 20;
        uint datatwo = 25;
        if(dataone<datatwo){
            return true;
            }
            else{
                return false;
            }
    }
}
```

03.03 Boolean Functions




```
pragma solidity ^0.5.0;

contract Operators {

    constructor() public {
    }

    function testNotEqual() public view returns(bool){
        uint dataone = 20;
        uint datatwo = 21;
        if(dataone!=datatwo){
            return true;
            }
            else{
                return false;
            }
    }

    function testGreater() public view returns(bool){
        uint dataone = 20;
        uint datatwo = 20;
        if(dataone>datatwo){
            return true;
            }
            else{
                return false;
            }
    }

    function testLessthan() public view returns(bool){
        uint dataone = 20;
        uint datatwo = 25;
        if(dataone<datatwo){
            return true;
            }
            else{
                return false;
            }
    }
}
```

SUMMARY OF LOGICAL CONDITIONAL OPERATORS

`A == B`	A EQUAL TO b
`A != B`	A NOT EQUAL TO b
`A > B`	A GREATER THAN B
`A < B`	A LESS THAN B
`A >= B`	A GREATER THAN OR EQUAL TO B
`A <= B`	A LESS THAN OR EQUAL TO B
`A && B`	BOTH A 'and' B ARE TRUE
`A || B`	EITHER A 'or' B IS TRUE
`!(A && B)`	BOTH A 'and' BARE FALSE

SUMMARY OF ARITHMETIC OPERATORS

A + B	ADDITION
A - B	SUBTRACTION
A * B	PRODUCT
A / B	QUOTIENT
A % B	MODULUS
A++	INCREMENT
A--	DECREMENT

SUMMARY OF ARITHMETIC ASSIGNMENTS

C = A + B	TRADITIONAL ARITHMETIC ASSIGNMENT
C += A	C = C + A
C -= A	C = C - A
C *= A	C = C * A
C /= A	C = C / A
C %= A	C = C % A

Programming in Solidity: PART 2

04 String, Arrays, and Structs

05 Mapping, Conversions, Special Variables

06 Functions, Modifiers, and Mathematical Functions

FILE EXPLORER

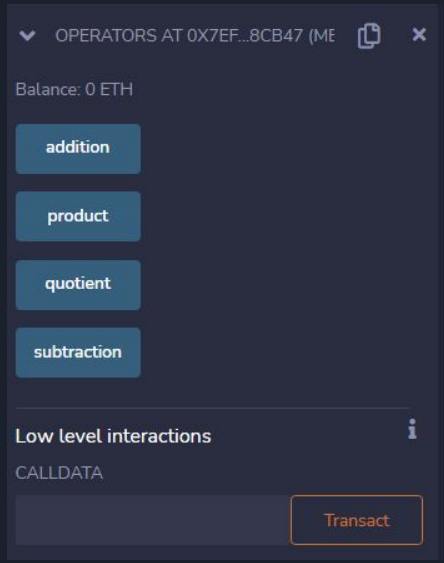
The default workspace in this environment is structured as follows:

/contracts

/scripts

Solidity Project 1

Interactive Calculator Application



Solidity Project 1: Part 2

Transfer ‘Ether’ from user to another

- Using the same concepts from earlier.
- Students will generate their own “ETH”
- Students will trade/ transfer their ETH between each other using their laptops.