भारतीय प्रौद्योगिकी संस्थान पटना
Indian Institute of Technology Patna

Course Code
CS577

Department of
Computer Science
and Engineering

Ethereum and Smart Contract Development in Solidity

By Fajge Akshay M.

Ethereum

- **world computer**
- **state transition system**

$$\sigma_{t+1} \equiv \Upsilon(\sigma_t, T)$$

Ethereum

- **world** computer

$$\sigma_{t+1} \equiv \Pi(\sigma_t, B)$$

$$B \equiv (\ldots, (T_0, T_1, \ldots))$$

$$\Pi(\sigma, B) \equiv \Omega(B, \Upsilon(\Upsilon(\sigma, T_0), T_1)\ldots)$$

Credit: Gavin Wood

Gas and Payment

To Avoid issue of network abuse

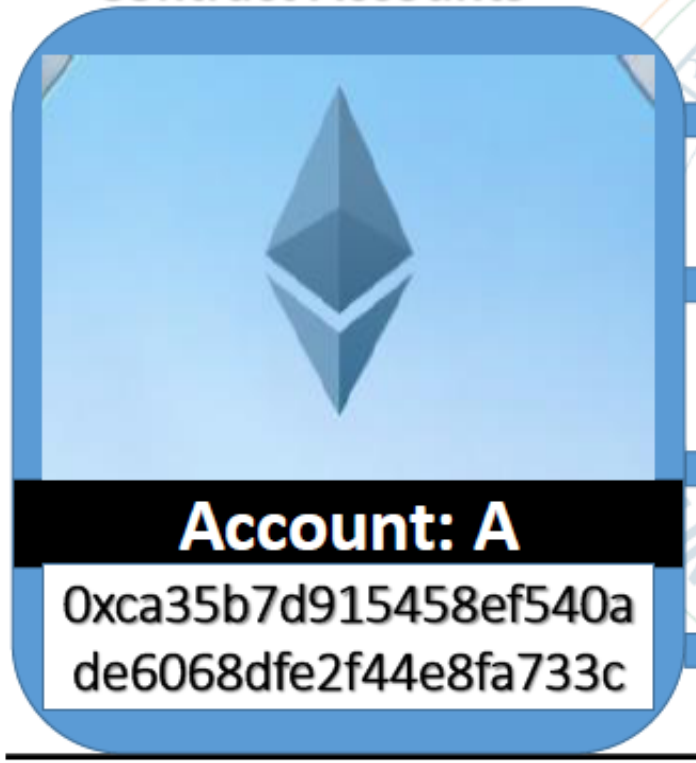
- Every transaction has a specific amount of gas associated with it

No mid-execution-refueling

Credit: Gavin Wood

Ethereum State: Accounts

- **Externally Owned Accounts**
- **Contract Accounts**



nonce => $\sigma[A]_n$ **|T| |Contracts|**

balance => $\sigma[A]_b$

storageRoot => $\sigma[A]_s$

codeHash => $\sigma[A]_c$ **"" H(code)**

Ethereum

- **From developers viewpoint:**
  - a global, open-source, publicly available, distributed platform for decentralized applications
- **From researchers perspective:**
  - set of consensus rules & algorithms, P2P network and communication protocols, EVM, Data structures, interoperable clients, **security protocols** and many more including **original specification and EIPs**
- **For a layman:**
  - It is a blockchain

Ethereum Client

- an application that implements the **Ethereum specification**
- implemented by **different developer groups** and in **different programming languages**
- **provides an interface** to interact with Ethereum
- communicates over the P2P network with other nodes
- **selection of an Ethereum client**
  - ✓development purpose
  - ✓deployment purpose

Ethereum Network

- Ethereum-based blockchain maintained by Ethereum clients
- **obey the formal specification** however they may or may not interoperate with each other
  - Mainnet
    - ✓the live public **Ethereum production blockchain**, where actual valued transactions take place on the distributed ledger visit: here
  - Testnet
    - ✓provides **platform to test software** before deploying on the Mainnet

Popular Ethereum Clients..

- **Geth (Go)**
- **Parity (Rust)**
- Webthree (C++)
- ruby-ethereum (Ruby)
- ethereumH (Haskell)
- ethereumj (Java)
- pyethereum (Python)
- ***Ganache/testrpc (Ethereumjs)***

many more…… check here

Ethereum Client: Geth

- “official” Ethereum client
- implemented in Go language implementation
- developed by Ethereum Foundation

- Download and Install
  - Geth: **here**

Setup: Truffle

1. Download and Install
   - GIT: here
   - Node.js: here (provides npm)
2. Setup Path variable for GIT and Node.js
3. Execute following commands::

   Execute following from an administrative PowerShell!
   - **npm install -g npm (npm Package Manager & Installer)**
   - **npm install -g windows-build-tools (install & configure the Visual Studio Build Tools and Python)**
   - **npm install -g ganache-cli truffle (** ethereumjs-testrpc along with truffle**)**

Ethereum Client: Ganache

- Download: here
- Provides personal Ethereum blockchain which you can use to run tests, execute commands, and inspect state while controlling how the chain operates.
  - Ganache (GUI)
  - Ganache develop
  - Ganache CLI

Geth: Useful commands and options

- **init** *…for Bootstrap and initialize*
- **account** *…for managing accounts*
  - **--unlock value** …comma separated list of accounts to unlock
- **console** *…for an interactive JavaScript environment*

- **--datadir** “<path>”
  …for data directory for the databases and keystore
- **--identity** <value> *…for providing custom node name*

Setup Private Blockchain Network

- Prerequisite
  - Ethereum client
  - Genesis block configuration file
  - Need to setup a network

Details will be covered in lab session

Introduction to Solidity

file: **HelloExample.sol**

```
1  //First Example.. Single line comment
2  pragma solidity 0.5.2;
3  //import './other_solidity_source_file.sol';
4  /*
5  * @title    HelloExample
6  * @author   Fajge Akshay M.
7  * @guide    Prof. Raju Halder
8  * @notice   Blockchain and Solidity tutorial for course CS577
9  */
10
11 /*
12 Multiline comments
13 reachus@email   fajge_1921cs12@iitp.ac.in; halder@iitp.ac.in
14 */
15
16 contract Hello {
17     function getMessage() public pure returns (string memory) {
18         return "Hello From Solidity";
19     }
20 }
```

- *Layout of source file*
  - contain an arbitrary number of
  - ✓ ***pragma directives***
  - ✓ ***import directives***
  - ✓ ***contract definitions***

Introduction to Remix IDE

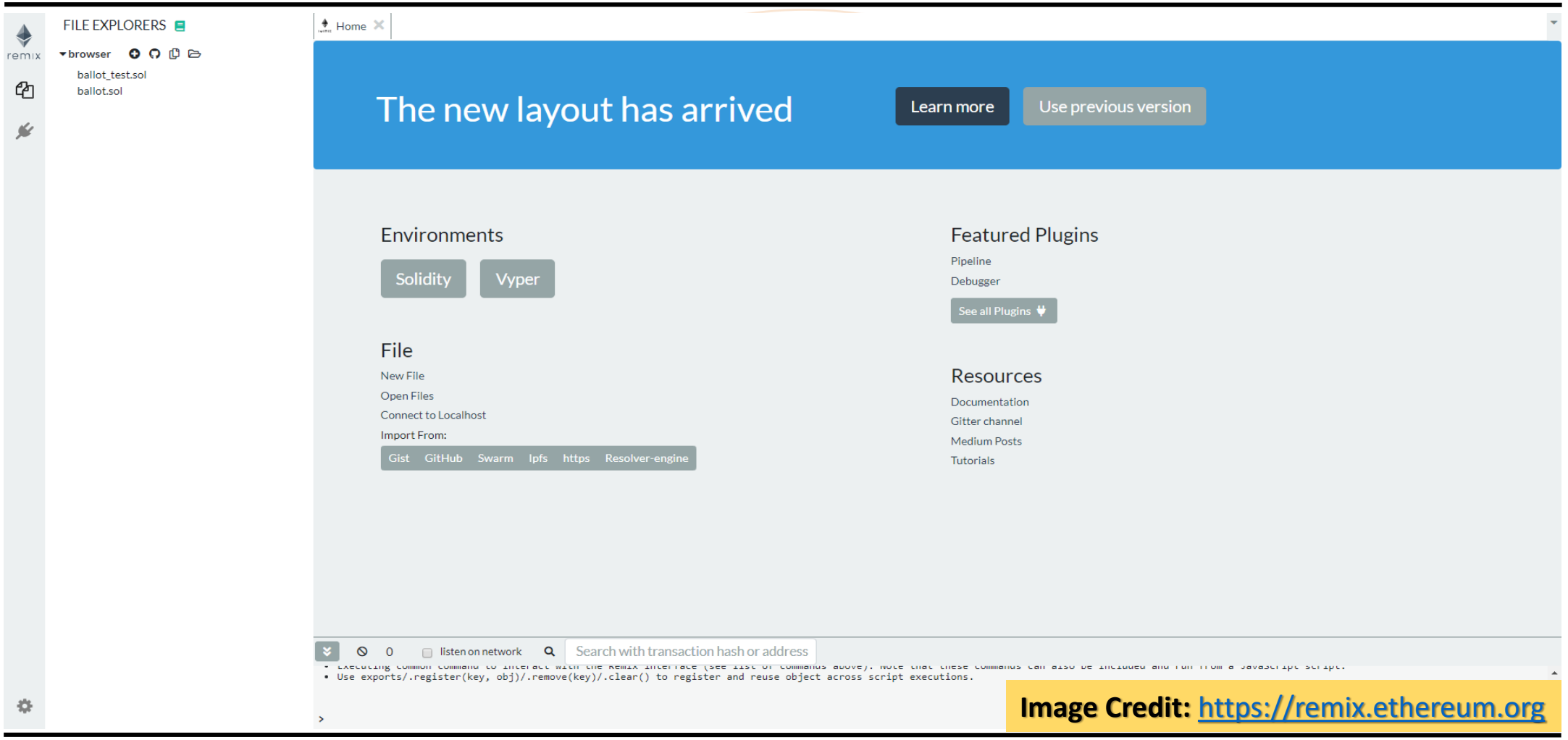


Image Credit: https://remix.ethereum.org

Fajge Akshay M. **Dr. Raju Halder**

Remix IDE: Plugin Manager

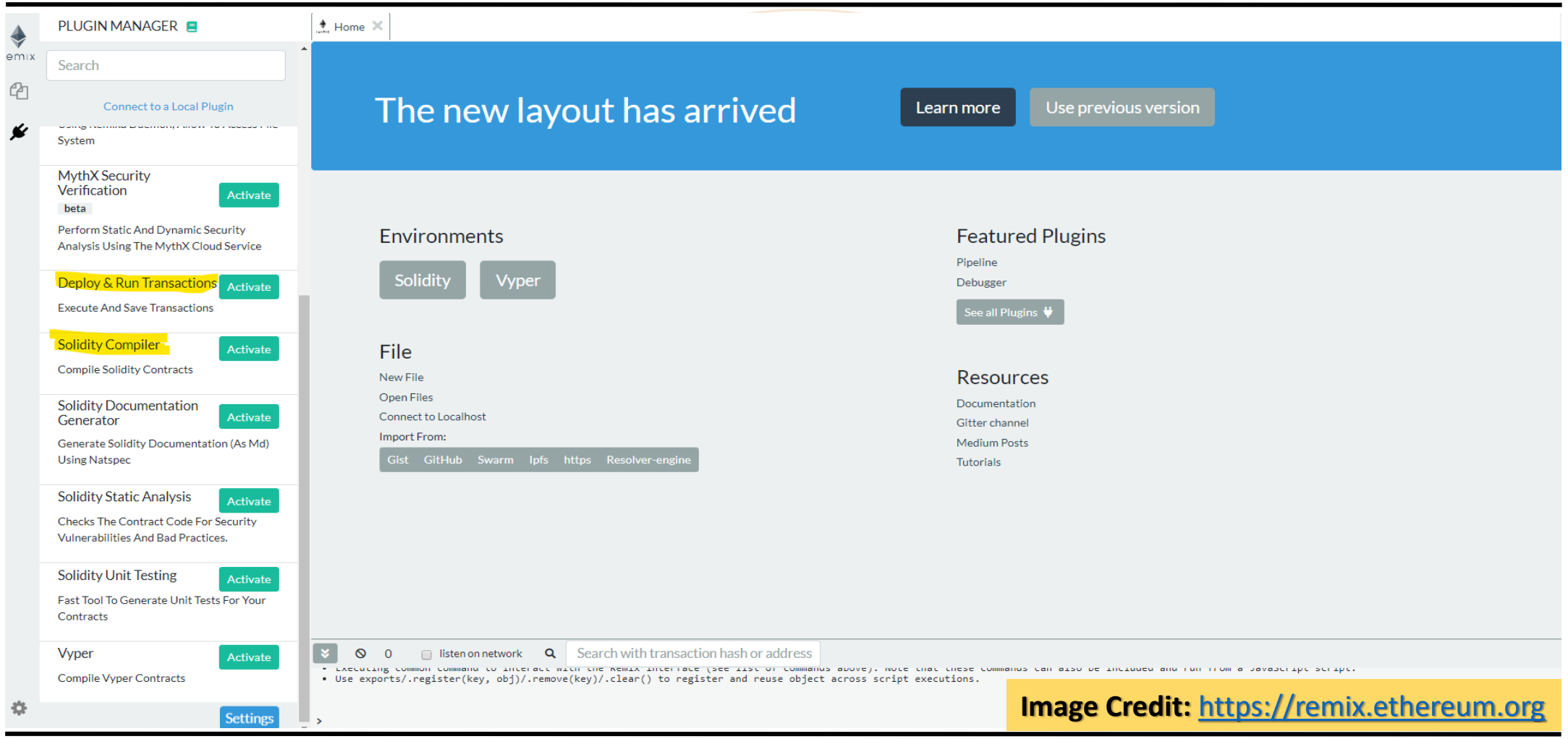


Image Credit: https://remix.ethereum.org

Fajge Akshay M.

Dr. Raju Halder

Solidity: Types

- Solidity is **statically typed** (like C, Java)
- **No concept** of "undefined", "Garbage" or "null" values
  - ✓ **Uninitialized variable holds default value based on the type of the variable**
- Support complex types (combination of elementary types) like struct

❑ **Different Types supported by Solidity**

- ✓ Value Types
- ✓ Contract Type
- ✓ Fixed-size and Dynamic-size byte arrays
- ✓ Enum Types
- ✓ Function Types
- ✓ Reference Types (struct, array, etc.)
- ✓ Mapping Types

Value Types

- **Booleans**
  keyword: bool **Literals:** true, false
- **Integers**
  Keyword: intX , uintX where X varies from 8 to 256 in steps of 8
  Literals: -4, 5, etc.
- **Fixed Point Numbers (fixedMxN , ufixedMxN) … *Not fully supported***
  M: the number of bits taken by the type
  N: the number of decimal points available must be between 0 and 80
- **Address**
  - Equivalent to size of Ethereum address i.e. 20 byte
  - Members of Address types:
    - ***balance, call, delegatecall, staticcall***

  Keyword: address, address payable
  - **address payable** supports additional members like ***transfer*** and ***send***

Other Types

- **Contract Type**
  - Keyword is the **name of already defined contract**

    ex: contract Hello …*if it is already defined then*

    Hello newHello = new Hello()
- **Fixed-size byte arrays**
  - **Keywords: bytesX** … where **X** varies from 1 to 32
  - Ex: bytes1, bytes2, …, bytes32
  - comes with Member "***length***"
- **Dynamic-size byte arrays or String Types**
  - **Keywords: string, bytes**

Few more types…

- **Enums**

  enum Feeling { Bored, Interested, Excited }

  Feeling student_feeling = Feeling.Bored

- **Function**

  function(<parameter types>) {public|internal|external|private} [pure|view|payable] [returns (<return types>)]

Reference Types

- **structs**, **arrays** and **mappings** types
- unlike Value Types, can have multiple reference variables
- need to provide data location explicitly

❑Data Locations

- ✓ **storage (location of state variables)**
- ✓ **memory (lifetime limited to an external function call)**
- ✓ **calldata (location of function arguments) -** only valid for parameters of external contract functions

Example: Struct, Enum

file: **StructExample.sol**

```
1  pragma solidity ^0.5.3;
2
3  contract StructExample {
4      enum PersonType { Faculty, Staff, Student}
5      struct Person {
6              string regNo;
7              address pAddress;
8              PersonType pType;
9      }
10     Person[] persons;
11
12     function register(string memory _regNo, PersonType _pType)
13     public returns(uint uid) {
14         uid = persons.length;
15         Person memory newPerson;
16         newPerson.regNo = _regNo;
17         newPerson.pType = _pType;
18         persons.push(newPerson);
19         return uid;
20     }
21
22     function getPersonType(uint uid)
23     external view returns (PersonType) {
24         return persons[uid].pType;
25     }
26 }
```

- *compiler version*
- *enum type*
- *struct type*
- *dynamic array*

- *How, when, and where "person" gets stored?*

Mapping Types

- **mapping**(Key => value)
- Key: any built-in Value Type including bytes and string
- Value: any Type including mapping type
- We can mark them as public

- In previous example to store persons we used Dynamic Array

Mapping vs Dynamic Array ?

Example: Mapping

file: **MappingExample.sol**

```
1  pragma solidity >=0.5.0 <0.6.0;
2
3  contract MapStructExample{
4      enum PersonType { Faculty, Staff, Student}
5      struct Person {
6              string regNo;
7              PersonType pType;
8      }
9
10     mapping (uint => Person) persons;
11     uint noOfPersons;
12
13     function register(string memory _regNo, PersonType _pType)
14     public returns(uint){
15         uint uid = noOfPersons;
16         Person memory newPerson;
17         newPerson.regNo = _regNo;
18         newPerson.pType = _pType;
19         persons[noOfPersons] = newPerson;
20         noOfPersons += 1;
21         return uid;
22     }
23
24     function getPersonType(int uid)
25     external view returns (PersonType) {
26         return persons[uint(uid)].pType;
27     }
28 }
```

- *compiler version*
- *enum type*
- *struct type*
- *mapping type*
- *array vs mapping*

? any difference

Global Namespace

❑ **Global Variables**

✓ Ether units: **wei**, **szabo** (1e12), **finney** (1e15), **ether** (1e18)

✓ Time units: **seconds**, **minutes**, **hours**, **days** and **weeks**

❑ **Special Variables and Functions**

- **Block and Transaction Properties**

  Ex: **msg.value**, **msg.sender**, **now**, **gasleft()**, **block.number**, etc...
- **ABI Encoding and Decoding Functions**
- **Error Handling**
- **Mathematical and Cryptographic Functions**

  addmod(uint a, uint b, uint m) *returns (uint)*

  keccak256(bytes memory) *returns (bytes32)*

Error Handling

- solidity uses **state-reverting** exceptions
- can throw error to the caller however no support for catching exceptions
- **Functions**
  - **assert(condition)** // for internal errors
  - **require(condition**, [errorMessageInString]**)** // for external components
    - ✓ **false** condition causes an invalid opcode which leads state change reversion
  - **revert(**[errorMessageInString]**)**

Example: Transfer

file: **TransferExample.sol**

```
1  pragma solidity 0.5.3;
2
3  contract TransferAmountExample{
4
5      function transferAmount(address payable recipientAccount, uint amount)
6      public
7      payable
8      {
9          recipientAccount.transfer(amount);
10     }
11
12     function getBalance(address account) public view returns(uint) {
13         return account.balance;
14     }
15 }
```

- **address payable** supports additional members like ***transfer*** and ***send***
- **Not a proper way…**

address payable?

payable?

What is the unit of the "amount"?

send vs transfer ?

Remix IDE: Deploy & Run

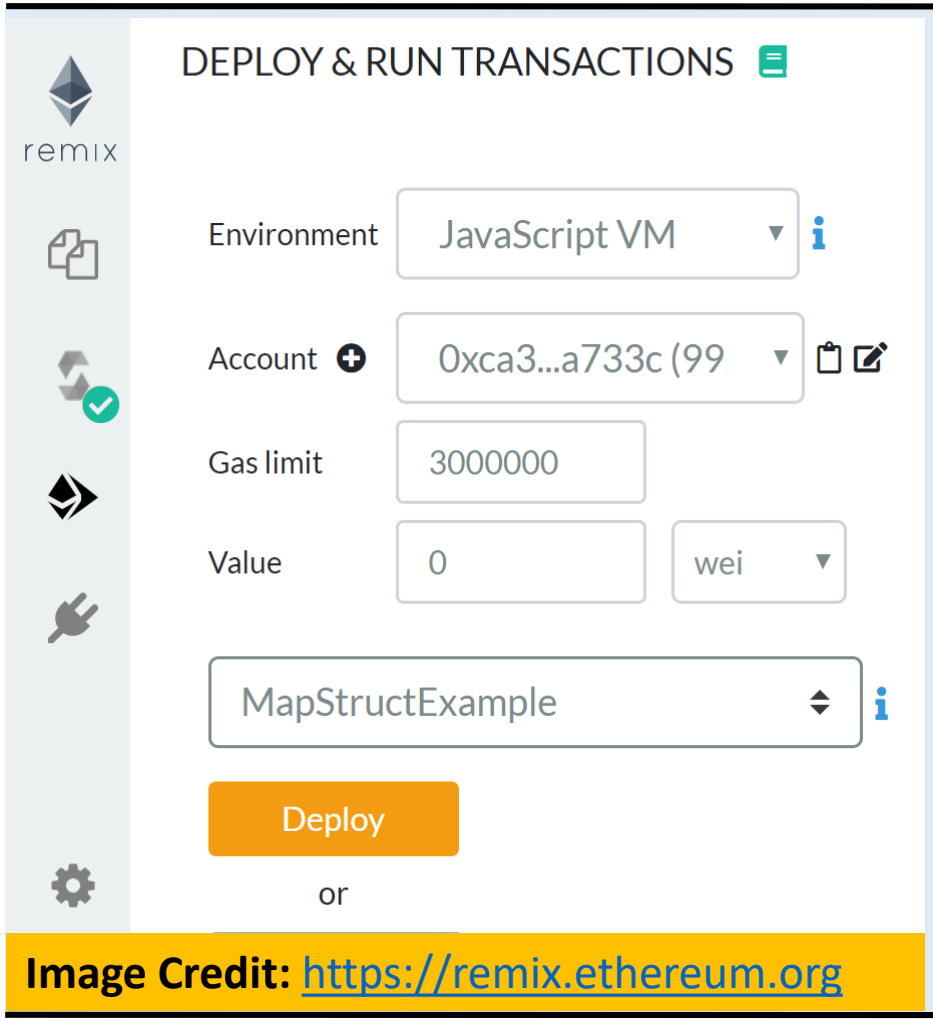


Image Credit: https://remix.ethereum.org

- **Environment**
- **Account**
- **Gas limit**
- **Value**
- **Compiled Class**
- **Deploy**

Example: require, assert

file: **ErrorHandlingExample.sol**

```
1  pragma solidity 0.5.3;
2
3  contract ErrorHandlingExample{
4
5      function transferAmount(address payable recipientAccount, uint amount)
6      public
7      payable {
8          recipientAccount.transfer(amount);
9      }
10
11     function getBalance(address account) public view returns(uint) {
12         return account.balance;
13     }
14
15     function modifiedTransferAmount(address payable accountAddr)
16     public payable returns (uint) {
17         uint avlBalance = msg.sender.balance;
18         require(msg.value <= avlBalance, "Insufficient Balance");
19         uint oldBalance = accountAddr.balance;
20         accountAddr.transfer(msg.value);
21         uint currBalance = msg.sender.balance;
22         assert(accountAddr.balance == oldBalance + msg.value);
23         assert(currBalance == avlBalance - msg.value);
24         return accountAddr.balance;
25     }
26 }
```

Previous transferAmount method?

Modified transferAmount method?

msg.sender
msg.value

Can we transfer 0 ethers?

Is code working?

- Any questions?
- Any issues with Remix?

Solution…

modified ErrorHandlingExample.sol

```
1  pragma solidity 0.5.3;
2
3  contract ErrorHandlingExample{
4
5      function transferAmount(address payable recipientAccount, uint amount)
6      public
7      payable {
8          recipientAccount.transfer(amount);
9      }
10
11     function getBalance(address account) public view returns(uint) {
12         return account.balance;
13     }
14
15     function modifiedTransferAmount(address payable accountAddr)
16     public payable returns (uint) {
17         uint avlBalance = msg.sender.balance;
18         require(msg.value <= avlBalance, "Insufficient Balance");
19         uint oldBalance = accountAddr.balance;
20         accountAddr.transfer(msg.value);
21         uint currBalance = msg.sender.balance;
22         assert(accountAddr.balance == oldBalance + msg.value);
23         //assert(currBalance == avlBalance - msg.value);
24         return accountAddr.balance;
25     }
26 }
```

require vs assert?

****Homework****

What is the issue with the line number 23?

Control Structures

- Different Control structures with semantics similar to C programming Language
  - ✓ **if, else, while, do, for, break, continue, return**

Except:

```
if (1921) {
    //body of If block
}
```

```
if (i = 45) {
    //body of If block
}
```

Example: ternary statement

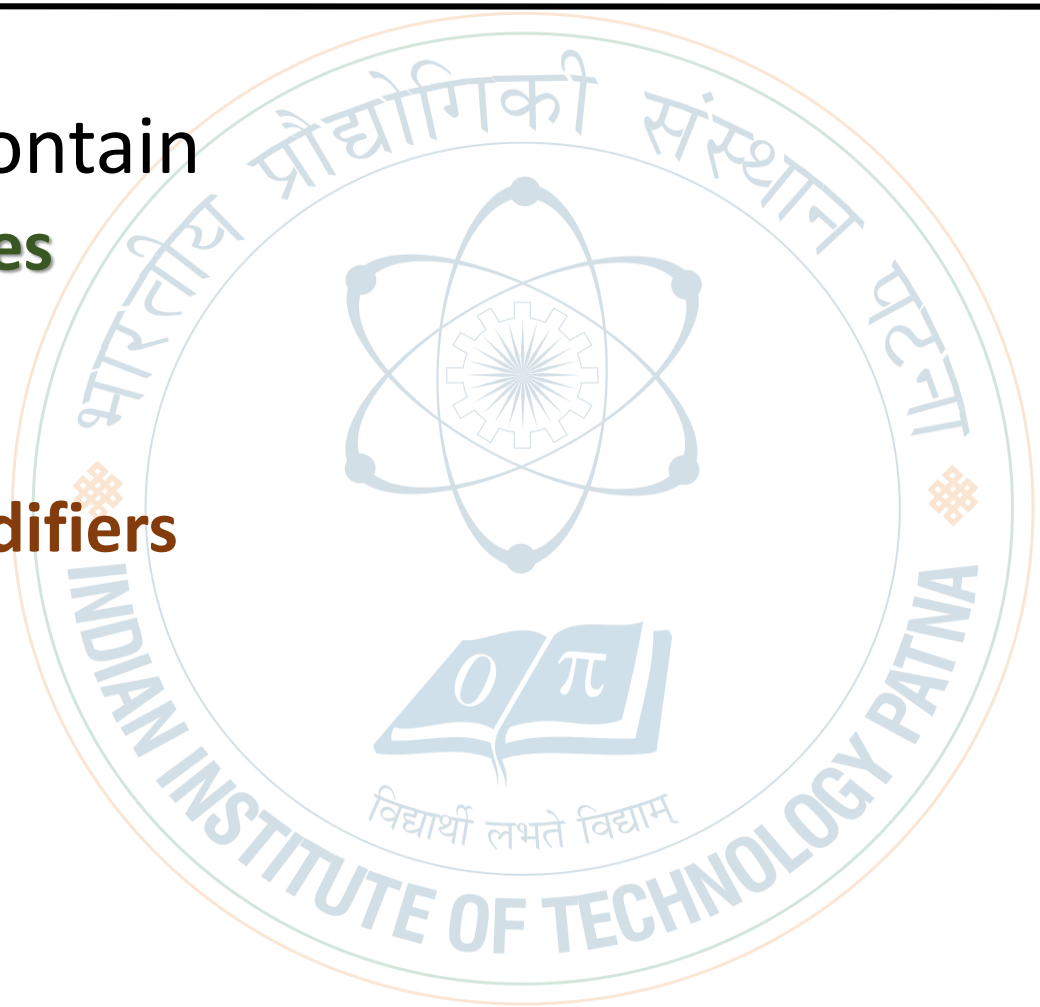
file: **Smallest.sol**

```
1  pragma solidity ^0.5.3;
2
3  contract ExampleDemo{
4
5      uint smallest = 2**256 - 1;
6      uint[] numbers;
7
8      function addNumber(uint number)
9      external
10     returns (uint) {
11             uint oldLength = numbers.length;
12             numbers.push(number);
13             smallest = number < smallest ? number : smallest;
14             assert(oldLength + 1 == numbers.length);
15             return smallest;
16     }
17
18     function getSmallest() public view returns(uint){
19         return smallest;
20     }
21
22 }
```

- *state variables*
- *Array members*
- *Function signature*
- *Ternary operator like C*
- *Assert*
- *view*

Structure of a Contract

- Contract can contain
  - ✓**State Variables**
  - ✓**Functions**
  - ✓**Constructor**
  - ✓**Function Modifiers**
  - ✓**Events**
  - ✓**Struct Types**
  - ✓**Enum Types**



Contracts

- Solidity source file can have multiple contracts
- **Special contracts**
  - **libraries**
  - **Interfaces**
- **Contract Creation**
  - using Ethereum transaction
  - using another contract

Constructor

file: **Constructor.sol**

```
1  pragma solidity 0.5.3;
2
3  contract ConstructorExample {
4      //AutoIncrementor
5      int256 private incrementBy;
6
7      constructor(int256 _incrementBy)
8      public {
9          incrementBy = _incrementBy;
10     }
11
12     function increment(int256 number)
13     public view returns(int256) {
14         return number + incrementBy;
15     }
16 }
```

- constructor is executed **after** contracts get created
- it is an **optional**
- **no** constructor **overloading**
- during contract creation process must know complete binary source code
- once constructor gets executed, code gets deployed on the network

Function

```
function(<parameter types>)
{private|public|internal|external}
[pure|view|payable]
[returns (<return types>)] {
// function body
}
```

- **Function overloading possible**
- **Function can return multiple values**
- **Function Visibility**
- **Function State Mutability**

Check this: Like C language, does it support block structures in function body?

Function Modifier

- frequently used to change behavior of the function
- gets executed prior to actual function call
- no-parameterized or parameterized
- multiple modifiers can be applied to constructor or functions
- code reusability

file: **Modifier.sol**

```
1  pragma solidity 0.5.3;
2
3  contract ModifierExample {
4      //AutoIncrementor
5      int256 private incrementBy;
6
7      modifier onlyPositive(int incrBy) {
8          require(incrBy > 0,
9          "_incrementBy must be > 0");
10         _;
11     }
12
13     constructor(int256 _incrementBy)
14     public onlyPositive(_incrementBy) {
15         incrementBy = _incrementBy;
16     }
17
18     function increment(int256 number)
19     public view returns(int256) {
20         return number + incrementBy;
21     }
22 }
```

Interesting Facts about Functions

- **no need of return statement explicitly** for the functions returning values
- all the variables declared in function parameters and return parameters gets **initialized to default values** based on type
- solidity provides **automatic getters** for the members with **public** access specifier
- can return multiple values

Example Demo: Revisited

Previous version: Smallest.sol

```
1  pragma solidity ^0.5.3;
2
3  contract ExampleDemo{
4
5      uint smallest = 2**256 - 1;
6      uint[] numbers;
7
8      function addNumber(uint number)
9      external
10     returns (uint) {
11             uint oldLength = numbers.length;
12             numbers.push(number);
13             smallest = number < smallest ? number : smallest;
14             assert(oldLength + 1 == numbers.length);
15             return smallest;
16     }
17
18     function getSmallest() public view returns(uint){
19         return smallest;
20     }
21
22 }
```

New Version: modify Smallest.sol

```
1  pragma solidity ^0.5.3;
2
3  contract ExampleDemo{
4      uint public smallest = 2**256 - 1;
5      uint[] numbers;
6
7      function addNumber(uint number)
8      external
9      returns (uint small) {
10             uint oldLength = numbers.length;
11             numbers.push(number);
12             small = number < smallest ? number : smallest;
13             smallest = small;
14             assert(oldLength + 1 == numbers.length);
15     }
16 }
```

Any changes in output?

Fallback Function

- anonymous function without parameters and return types
- **visibility:** must be **external**
- **how do we make a call?**
- gets executed **only when called function which is not existed in the contract**

Events

- an abstraction on top of the EVM's logging functionality
- on an event call, arguments to be stored in the transaction's log
- log and its event data is not accessible from within contracts
- events are emitted using **emit**

Inheritance

- keyword used: **is**
- supports **multiple inheritance** including **polymorphism**
- function calls are virtual (**except** they called using ***contract_name*** or ***super***)
- like object-oriented programming, only one contract gets created and deployed

Inheritance

- supports **multiple inheritance** including **polymorphism**
- **inappropriate** multiple inheritance may lead to **Type error**
- **linearization of inheritance graph** must be considered

- **what will if inheriting result in different types of members with same identifier?**

Abstract Contract

- contract must have at least one function without implementation
- can be used as a base contract
- can not compile

Interface

- similar to abstract contracts, but without any functions definition
- cannot inherit other contracts or interfaces
- visibility must be external
- no state variables and constructor
- **only Enums, Structs, Functions**

Example:

file: **Inheritance.sol**

```
1  pragma solidity >=0.5.2 <0.5.4;
2
3  contract Person {
4      function does()
5      public returns (string memory);
6  }
7
8  contract Info {
9      string name;
10     constructor(string memory _name) internal{
11         name = _name;
12     }
13
14     function getName()
15     public view returns (string memory) {
16         return name;
17     }
18 }
19
20 contract Student is Person, Info {
21     function does()
22     public returns (string memory) {
23         return "Learning";
24     }
25     constructor(string memory _name)
26     Info(_name) public{
27     }
28 }
```

```
29
30 contract Faculty is Info, Person {
31     string name = "Dr. Raju Halder";
32
33     function does()
34     public returns (string memory) {
35         return "Teaching";
36
37     }
38     constructor() Info(name) public{
39
40     }
41 }
42
43 contract TA is Person, Info("Fajge Akshay") {
44     function does()
45     public returns (string memory) {
46         return "helps Teachers and Studetns";
47     }
48 }
```

Example

```
49
50  contract Test{
51      TA ta = new TA();
52      Faculty faculty = new Faculty();
53      Student student = new Student('Your Name');
54
55      function getTA() public view returns (string memory){
56          return ta.getName();
57      }
58
59      function getFaculty() public view returns (string memory){
60          return faculty.getName();
61      }
62
63      function getStudent() public view returns (string memory){
64          return student.getName();
65      }
66  }
```

Deploy Test contract and verify result…

Change *Person* abstract class as given below to treat it as interface:

```
1  pragma solidity >=0.5.2 <0.5.4;
2
3  interface Person {
4      function does()
5      external returns (string memory);
6  }
```

Is it working?

Libraries

- internal functions of library are visible in all contracts
- worked as an implicit base class, however no trace in hierarchy
- deploy once and use anywhere like base class, library code gets executed in callers context
- improves code reusability

Example: library

file: **myLib.sol**

```
1  pragma solidity 0.5.3;
2  library Utils {
3      function arrayContains
4      (uint256[] memory input, uint256 value)
5      internal pure returns (bool flag){
6          flag = false;
7          for(uint i = 0; i<input.length; i++){
8              if (input[i]==value)
9                  return true;
10         }
11     }
12 }
```

Example: import

file: **importExample.sol**

```
1  pragma solidity 0.5.3;
2  import './myLib.sol' as Akshay;
3  contract ModifiedSmallestDemo {
4      uint public smallest = 2**256 - 1;
5      uint[] numbers;
6
7      function addNumber(uint number)
8      external
9      returns (uint small) {
10         uint oldLength = numbers.length;
11         numbers.push(number);
12         small = number < smallest ? number :  smallest;
13         smallest = small;
14         assert(oldLength + 1 == numbers.length);
15     }
16
17     function contains(uint value)
18     external view returns(bool){
19         return Akshay.Utils.arrayContains(numbers, value);
20     }
21 }
```

Source Code of Examples:

SCAN

or

Visit here

Ethereum Development: Language

- **Smart Contract**
  - ✓ **Requires knowledge of how to write Smart Contracts**

  - **Languages:**
    - **Solidity (Javascript based)**
    - **Vyper (Pythonic programming language)**

  - **IDEs:**
    - **Remix - Ethereum IDE (for beginners) online editor: here**
    - **Visual Studio Code (extension:: juanblanco.solidity)**
    - **Eclipse (extension:: YAKINDU Solidity Tools)**
    - **IntelliJ-Solidity**
    - **Atom**

Ethereum Development: Network

- **Mainnet**
  - ✓the live public Ethereum production blockchain, where actual valued transactions take place on the distributed ledger. **visit:** here
- **Testnet**
  - ✓**Provides platform to test software before deploying on the Mainnet**
  - **Local Testnets**
    - **Ganache**
    - **Ganache CLI**
  - **Public Testnets** ***(ethers requested via faucet)***
    - **Ropsten (Geth/Parity) (Minable)**
    - **Görli (Multi-client)**
    - **Kovan (Parity)**
    - **Rinkeby (Geth)**

Ethereum Development: Interface

- **Front-end interface**
  - **Web3.js (Ethereum JavaScript API)**
    - collection of libraries which allow you to interact with a local or remote Ethereum node, using a HTTP or IPC connection. **(documentation: here)**
  - **Ethers.js**
    - a complete and compact library for interacting with the Ethereum Blockchain and its ecosystem along with Ethereum wallet implementation **(documentation: here)**
- **Back-end interface**
  - **Web3j**
    - a lightweight, reactive, type safe library for Java, Android, Kotlin and Scala to connect with Ethereum clients **(documentation: here)**

Ethereum Development

- **Testing and Deployment:**
  - **Truffle suite, Embark, Waffle**

- **Storage:**
  - **IPFS**
  - **Swarm**

Happy Learning!

Thank you!

For Examples

Scan Me

Fajge Akshay M.

Dr. Raju Halder