OST
Eastern Switzerland
University of Applied Sciences

# Blockchain (BlCh)

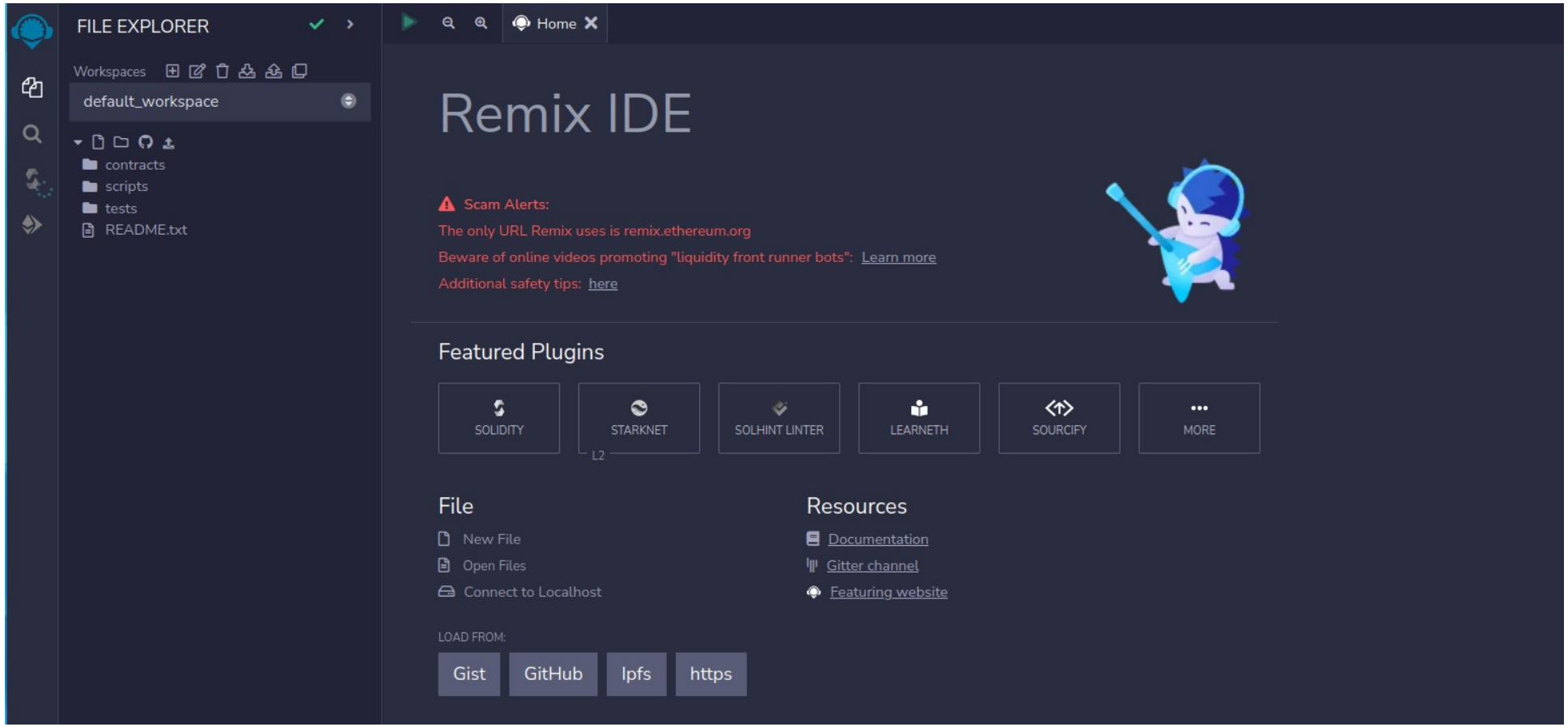
## Solidity

Thomas Bocek

30.09.2024

# Solidity IDE

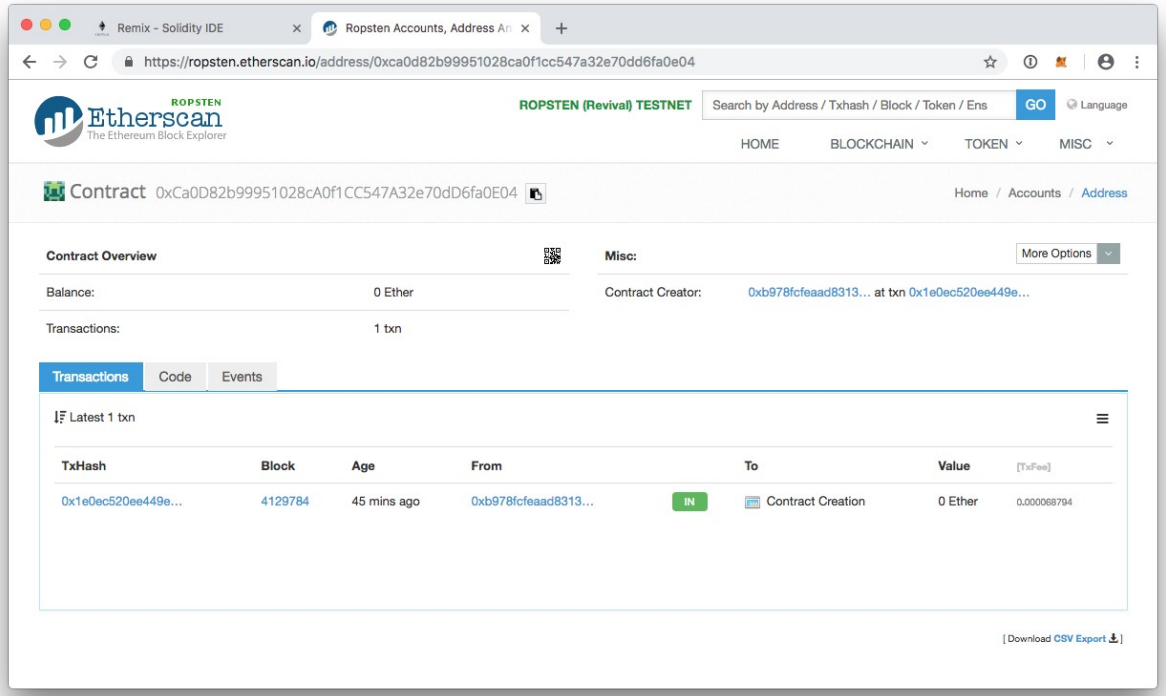
- https://remix.ethereum.org
  - IDE e.g., in combination with Solidity Intellij plugin (not ideal)




OST

# Check Deployment

- Etherscan is a public blockchain explorer
  - Shows all transactions, blocks, understands tokens, also works for testnets
  - Mine you first contract! 🎉




OST

# Gas: Ethereum's Fuel

- Price that is paid for running a transaction or a contract in the Ethereum VM (EVM)
- EVM: can execute instructions (opcodes) → yellow paper (on the right)
- Unit of measuring computational work
- Every instruction needs to be paid for
- If you run out of gas, state is reverted, ETH gone

APPENDIX G. FEE SCHEDULE

The fee schedule $G$ is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

| Name | Value | Description* |
|---|---|---|
| $G_{zero}$ | 0 | Nothing paid for operations of the set $W_{zero}$. |
| $G_{base}$ | 2 | Amount of gas to pay for operations of the set $W_{base}$. |
| $G_{verylow}$ | 3 | Amount of gas to pay for operations of the set $W_{verylow}$. |
| $G_{low}$ | 5 | Amount of gas to pay for operations of the set $W_{low}$. |
| $G_{mid}$ | 8 | Amount of gas to pay for operations of the set $W_{mid}$. |
| $G_{high}$ | 10 | Amount of gas to pay for operations of the set $W_{high}$. |
| $G_{extcode}$ | 700 | Amount of gas to pay for operations of the set $W_{extcode}$. |
| $G_{balance}$ | 400 | Amount of gas to pay for a BALANCE operation. |
| $G_{sload}$ | 200 | Paid for a SLOAD operation. |
| $G_{jumpdest}$ | 1 | Paid for a JUMPDEST operation. |
| $G_{sset}$ | 20000 | Paid for an SSTORE operation when the storage value is set to non-zero from zero. |
| $G_{sreset}$ | 5000 | Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero. |
| $R_{sclear}$ | 15000 | Refund given (added into refund counter) when the storage value is set to zero from non-zero. |
| $R_{suicide}$ | 24000 | Refund given (added into refund counter) for suiciding an account. |
| $G_{suicide}$ | 5000 | Amount of gas to pay for a SUICIDE operation. |
| $G_{create}$ | 32000 | Paid for a CREATE operation. |
| $G_{codedeposit}$ | 200 | Paid per byte for a CREATE operation to succeed in placing code into state. |
| $G_{call}$ | 700 | Paid for a CALL operation. |
| $G_{callvalue}$ | 9000 | Paid for a non-zero value transfer as part of the CALL operation. |
| $G_{callstipend}$ | 2300 | A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer. |
| $G_{newaccount}$ | 25000 | Paid for a CALL or SUICIDE operation which creates an account. |
| $G_{exp}$ | 10 | Partial payment for an EXP operation. |
| $G_{expbyte}$ | 10 | Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation. |
| $G_{memory}$ | 3 | Paid for every additional word when expanding memory. |
| $G_{txcreate}$ | 32000 | Paid by all contract-creating transactions after the *Homestead transition*. |
| $G_{txdatazero}$ | 4 | Paid for every zero byte of data or code for a transaction. |
| $G_{txdatanonzero}$ | 68 | Paid for every non-zero byte of data or code for a transaction. |
| $G_{transaction}$ | 21000 | Paid for every transaction. |
| $G_{log}$ | 375 | Partial payment for a LOG operation. |
| $G_{logdata}$ | 8 | Paid for each byte in a LOG operation's data. |
| $G_{logtopic}$ | 375 | Paid for each topic of a LOG operation. |
| $G_{sha3}$ | 30 | Paid for each SHA3 operation. |
| $G_{sha3word}$ | 6 | Paid for each word (rounded up) for input data to a SHA3 operation. |
| $G_{copy}$ | 3 | Partial payment for *COPY operations, multiplied by words copied, rounded up. |
| $G_{blockhash}$ | 20 | Payment for BLOCKHASH operation. |

$W_{zero} = \{\text{STOP, RETURN}\}$
$W_{base} = \{\text{ADDRESS, ORIGIN, CALLER, CALLVALUE, CALLDATASIZE, CODESIZE, GASPRICE, COINBASE, TIMESTAMP, NUMBER, DIFFICULTY, GASLIMIT, POP, PC, MSIZE, GAS}\}$
$W_{verylow} = \{\text{ADD, SUB, NOT, LT, GT, SLT, SGT, EQ, ISZERO, AND, OR, XOR, BYTE, CALLDATALOAD, MLOAD, MSTORE, MSTORE8, PUSH*, DUP*, SWAP*}\}$
$W_{low} = \{\text{MUL, DIV, SDIV, MOD, SMOD, SIGNEXTEND}\}$
$W_{mid} = \{\text{ADDMOD, MULMOD, JUMP}\}$
$W_{high} = \{\text{JUMPI}\}$
$W_{extcode} = \{\text{EXTCODESIZE}\}$

# Solidity Source File

- SPDX License Identifier
  - List, example: // SPDX-License-Identifier: MIT
  - Private code: UNLICENSED
- Version Pragma
  - ```
    pragma solidity ^0.4.24;
    // not before 0.4.24, before 0.5.0
    ```
  - ```
    pragma solidity >=0.4.22 <0.9.0;
    // not before 0.4.22, before 0.9.0
    ```
- Importing
  - ```
    import "filename";
    //into the current global scope
    ```
- Comments
  - ```
    // This is a single-line
    comment.
    ```
  - ```
    /*
    This is a
    multi-line comment.
    */
    ```

OST
    ```

# Solidity Structure

- State variables
  - Stores state persistently, expensive to write!

```
contract SimpleStorage {
    uint256 storedData;
    // state variable
}
```

- Functions
  - Internal or external calls
  - `function bid() public {`

```
  // ...
}
```

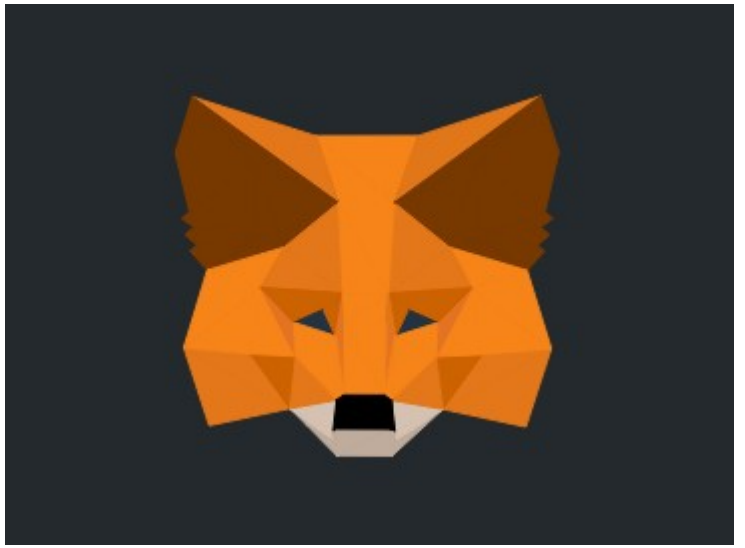
- Visibility
  - Specify from where functions can be called
    - Internal / private: only callable internally
      - Internal: can be overridden, private not
    - External: only callable from outside
    - Public: callable from internally / outside
- Types
  - pure
    - No state read or write
  - view
    - No state write but state read
  - payable
    - Can send or receive ETH

OST

# Solidity IDE

- Create a first contract

```
pragma solidity ^0.8.21;
// Minimal contract example
contract SimpleStorage {
    uint256 storedData; // State variable
}
```



- Install MetaMask
- Compile
- Compile and push «Deploy»

OST

# Solidity Structure

- Modifiers (e.g., OpenZeppelin)
  - Called before the function, e.g.,
  - ```
    modifier onlyOwner() {
      require(
        msg.sender == owner, "Only
    owner allowed");
      _;
    }
    ```
  - ```
    function abort() public view
    onlyOwner{
            // …
    }
    ```
- Modifier / Function Overriding
  - Functions can be overridden → virtual
  - Function that overrides → override
- Example
  - function test() public view virtual returns (bool)

OST

# Solidity Structure

- Events
  - Communicate to (not from!) the outside
  - `event HighestBidIncreased(string msg);`

```
function bid() public payable {
  // ...
  emit HighestBidIncreased(“hallo”);
}
```

  - Sometimes misused for debugging (hint: use Hardhat, console.log)

- Errors, use with revert
  - `error NotEnoughFunds(uint requested);`

```
function transfer(address to) public {
if (balance < amount)
  revert NotEnoughFunds(amount);
}
```

  - Often require is used, but more expensive
  - `require(balance >= amount, "Not enough");`
  - Not used that often: `assert`
    - For catching bugs in your contract
  - `try/catch` also supported, not for assert, but for require/revert

OST

# Solidity – Events/Notifications

- Events are a way for smart contracts written in Solidity to log that something has occurred
- Interested observers, notably JavaScript front ends for decentralized apps, can watch for events and react accordingly.

# Solidity Structure

- Struct
  - Define custom types
  - ```
    contract Ballot {
        struct Voter {
            uint weight;
            bool voted;
            address delegate;
            uint vote;
        }
    }
    ```
- Enum
  - Custom types with a set of ‘constant values’
  - ```
    contract Purchase {
        enum State { Created, Locked, Inactive }
    }
    ```

OST

# Solidity Types and Operators

- Boolean
  - ! (logical negation)
  - && (logical conjunction, “and”)
  - || (logical disjunction, “or”)
  - == (equality)
  - != (inequality)
  - No short-circuit evaluation
    - Probably you should not use it in other languages as well (my opinion)
- Integers
  - int/uint from 8 to 256 bit → e.g, uint256
    - Comparisons: <=, <, ==, !=, >=, >
    - Bit operators: &, |, ^ (exclusive or), ~ (negation)
    - Shift operators: << (left shift), >> (right shift)
    - Arithmetic operators: +, -, unary - (only for signed integers), *, /, % (modulo), ** (exponentiation)
  - Not yet ready: fixed / ufixed → ufixed128x18 (18 decimal points)
- Address
  - `address / address payable`
    - `balance`
    - `transfer`
    - `send`
    - `call, delegatecall and staticcall` - typically use: `ERC721(address).balanceOf(...)`

OST

# Solidity Types and Operators

- Arrays
  - Fixed size or dynamic (slices :)
  - `bytes1 … bytes32`
  - `pop/push/length`
- User-defined Value Types
  - Rarely used in simple contracts
- Data Location
  - storage: often copy
  - memory: references

- Mapping ~hash table (without iteration)
  - ```
    mapping(address => uint) public balances;
    function update(uint newBalance) public {
      balances[msg.sender] = newBalance;
    }
    ```
- Ternary Operator: ?
- Constant / immutable
- Using statement
  - used often before Solidity 0.8 in SafeMath
  - `uint256 amount1 = amount1.sub(amount2);`
  - Now: `uint256 amount1 -= amount2;`

OST
    ```

# Units and Builtin Variables / Control Structures

- Complete list [link]
  - wei, gwei or ether
  - seconds, minutes, hours, days and weeks
  - blockhash, blocknumber
  - block.prevrandao (**new**)
  - block.timestamp
  - msg.data
  - **msg.sender**
  - **msg.value**

- `if, else, while, do, for, break, continue, return`
- Creating Contracts
  - `new` within contract, `web3.eth.Contract` from outside
  - Constructor
- Inheritance
  - Base contracts from OpenZeppelin
  - `contract ERC721 is Context, ERC165, IERC721, IERC721Metadata`
  - `abstract contract / interface`

OST

# Control Structures

- Can a contract deploy another contract?
  - Yes
  - ```
    contract ChildContract {
        string public data;
        constructor(string memory _data) {
            data = _data;
        }
    }

    contract FactoryContract {
        // address of the last deployed ChildContract
        address public lastDeployedAddress;
        function deployChild(string memory _data) public {
            // Deploy a new instance of ChildContract
            ChildContract child = new ChildContract(_data);
            // Store the address of the deployed contract
            lastDeployedAddress = address(child);
        }
    }
    ```

- You probably don't need this: Inline Assembly
  - ```
    assembly {
      // retrieve the size of
      //the code, this needs
      //assembly

      let size :=
        extcodesize(addr)

    }
    ```
- `unchecked{}` → make variables under/overflow
  - Used to optimize gas usage

OST

# Many References / Tutorials

- https://consensys.github.io/smart-contract-best-practices/
- https://learnxinyminutes.com/docs/solidity/
- https://consensys.net/blog/developers/solidity-best-practices-for-smart-contract-security/
- https://www.tutorialspoint.com/solidity/solidity_basic_syntax.htm
- https://docs.soliditylang.org/en/v0.8.27/
- https://www.dappuniversity.com/articles/solidity-tutorial
- https://www.w3schools.io/blockchain/solidity-tutorials/

OST