# CST4125: Blockchain Development
Week: 18
Title: Security & Unit Testing

Dr Ian Mitchell


Bedfordshire,
UK

2023



## Contact and Office Hours

### Contact Details

- Name: Dr Ian Mitchell
- Room: TG10
- Address: Middlesex University, Computer Science, London, NW4 4BT
- email: smerf.net



## Contact and Office Hours

### Contact Details

- Name: Dr Ian Mitchell
- Room: TG10
- Address: Middlesex University, Computer Science, London, NW4 4BT
- email: smerf.net

### Office Hours

- During term time only
- When: Autumn Term: Mondays 1100-1300hrs
- Please read notifications or emails
- There are occassions that these could be arranged online, e.g., due to industrial action or inclement weather



## Deadlines

| | | | | Feedback | |
|---|---|---|---|---|---|
| Description | Submission | Weight | Deadline | Formative | Summative |
| 1. Hyperledger | MyLearning | 50% | 18th December 2022 | LW11-12 | 12/01/2023 |
| 2. Ethereum | MyLearning | 50% | 2nd April 2023 | LW23-24 | 24/04/2023 |
| Resits | MyLearning | 50-100% | 1st July 2023 | None | None |
| Deferals | MyLearning | 50-100% | 1st July 2023 | None | None |



## Lecture Objectives

- Private Networks
- Truffle, Ganache, node, geth
- Local remix
- Unit testing
- Functions
- Events & Logging
- Debugging
- Etherscan



## Error Handling

- Inadvertent
- Robust contracts
- Run-time

## Error Handling

- Inadvertent
- Robust contracts
- Run-time
  - out-of-gas errors
  - divide by zero
  - data overflow
  - array errors
- `throw` is now obsolete
- no `try ... catch`
- `require, revert, assert`



## require

- prerequisites
- conditions
- `require(` $<$ *condition* $>$`)`
- evaluates as boolean
- declare conditions that are satisfied before executing code



## require

```
    uint32 constant max=8000;
    uint32 constant min=1000;
    event lessThan(string);
    event greaterThan(string);
    event inRange(string);
    function lessThanX(uint32 _x) public {
        require(_x <= max);
        emit lessThan("Integer is less than");
    }
    function greaterThanX(uint32 _x) public  {
        require(_x >= min);
        emit greaterThan("Integer is greater than");
    }
    function inRangeFn(uint32 _x) public  {
        require(_x>=min);
        require(_x<=max);
        emit inRange("Integer is in Range");
    }
}
```



## require with help

```
contract test{
    uint32 constant max=8000;
    uint32 constant min=1000;
    event lessThan(string);
    event greaterThan(string);
    event inRange(string);
    function lessThanX(uint32 _x) public {
        require(_x <= max, "less than 8000");
        emit lessThan("Integer is less than");
    }
    function greaterThanX(uint32 _x) public  {
        require(_x >= min, "greater than 1000");
        emit greaterThan("Integer is greater than");
    }
    function inRangeFn(uint32 _x) public  {
        require(_x>=min, " > 1000");
        require(_x<=max, " < 8000");
        emit inRange("Integer is in Range");
    }
}
```



## Gate condition

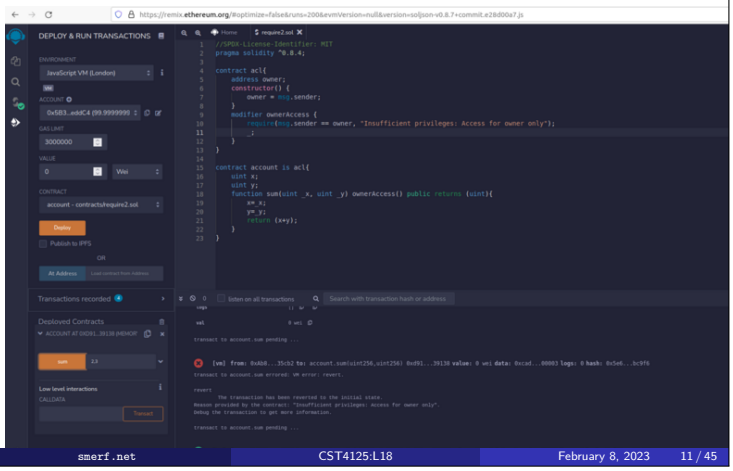
- Make some access control rules
- inherit them
- include them as modifiers
- executed in order of modifier



## Access Modifiers

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;
contract acl{
    address owner;
    constructor() {
        owner = msg.sender;
    }
    modifier ownerAccess {
        require(msg.sender == owner, "Insufficient privileges: Access for owner only");
        _;
    }
}
contract account is acl{
    uint x;
    uint y;
    function sum(uint _x, uint _y) ownerAccess() public returns (uint){
        x=_x;
        y=_y;
        return (x+y);
    }
}
```

## Access Modifiers






## assert

- same syntax as `require` statement
- does not accept an error message as a parameter (1 parameter)
- accepts a condition that either evaluates to `true` or `false`
- If it is true then executes code, else throws exception
- When to assert?
- `require`: values from outside
- `assert`: state and condition of the function
- current state has become inconsistent
- expected outcome is true



## Assert Example

```
1  //SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.4;
3
4  contract test{
5      uint32 constant max=8000;
6      uint32 constant min=1000;
7      event lessThan(string);
8      event greaterThan(string);
9      event inRange(string);
10     function sum(uint32 _x, uint32 _y) public returns (uint32){
11         if (inRangeFn(_x) && inRangeFn(_y)){
12             assert(_x+_y<=max);
13             return uint32(_x+_y);
14         }
15     }
16     function inRangeFn(uint32 _x) public returns (bool) {
17         require(_x>=min, " > 1000");
18         require(_x<=max, " < 8000");
19         return (true);
20         emit inRange("Integer is in Range");
21     }
22     function lessThanX(uint32 _x) public {
23         require(_x <= max, "less than 8000");
24         emit lessThan("Integer is less than");
25     }
26     function greaterThanX(uint32 _x) public  {
27         require(_x >= min, "greater than 1000");
28         emit greaterThan("Integer is greater than");
29     }
30
31
32 }
```



## revert

- no parameters or conditions
- `revert` throws an exception
- returns unused gas
- reverts to original state
- similar to `require` without conditions



## revert Example

```
1  //SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.4;
3  contract testRevert{
4      function checkUint8(uint16 _x) public pure returns(bool){
5          if( _x > 255){
6              revert();
7          }
8          return (true);
9      }
10 }
```



## Testing

- Testing increases gas consumption
- right balance
- use of comments
- `testnet` err on the side of caution
- `mainnet` be frugal, but comment

## Truffle

- Open terminal in VM
- Create a directory, and move there
- `mkdir truffleProject && cd truffleProject`
- `truffle init`
- Then open the project in VSCode
- `code .`
- Compile
- Migrate
- Test



## Contract

```
1 //SPDX-License-Identifier:GPL3.0
2 pragma solidity ^0.8.0;
3 contract myFirstContract{
4     uint8 sum;
5     function add(uint8 _x, uint8 _y) public returns(uint8){
6         require(_x+_y<255);
7         return uint8(_x+_y);
8     }
9 }
```

Figure: Listing for `myFirstContract.sol`.



## Configuration file

```
37   networks: {
38     // Useful for testing. The 'development' name is special - truffle uses it by default
39     // if it's defined here and no other network is specified at the command line.
40     // You should run a client (like ganache, geth, or parity) in a separate terminal
41     // tab if you use this network and you must also set the 'host', 'port' and '
        network_id'
42     // options below to some value.
43     //
44     development: {
45       host: "127.0.0.1",     // Localhost (default: none)
46       port: 7545,            // Standard Ethereum port (default: none)
47       network_id: "*",       // Any network (default: none)
48     },
```

Figure: Partial Listing for `truffle-config.js`.



## Contract Configuration

```
1 const myFirstContract = artifacts.require("myFirstContract");
2 module.exports = function (deployer) {
3   deployer.deploy(myFirstContract);
4 };
```

Figure: Listing for `2_myFirstContract.js`. Notice filenaming convention



## Project Structure

```
1  .
2  |-- build
3  |   `-- contracts
4  |       |-- Migrations.json
5  |       `-- myFirstContract.json
6  |-- contracts
7  |   |-- Migrations.sol
8  |   `-- myFirstContract.sol
9  |-- migrations
10 |   |-- 1_initial_migration.js
11 |   `-- 2_myFirstContract.js
12 |-- test
13 |-- tree.lst
14 `-- truffle-config.js
15
16 5 directories, 8 files
```

Figure: Directory and File structure of an Ethereum project



## Compile and Migrate

1. `truffle compile`
2. `truffle migrate`

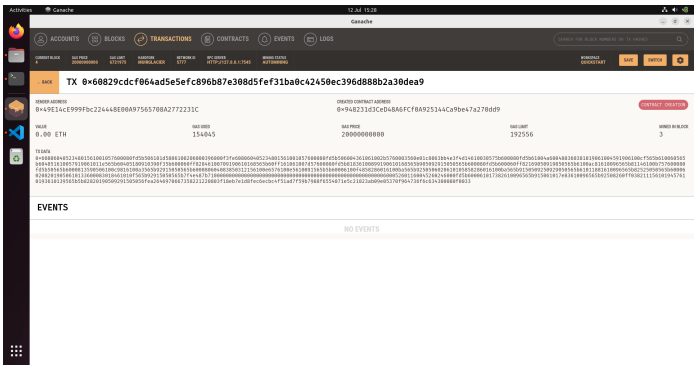
## Compile and Migrate

1. `truffle compile`
2. `truffle migrate`
3. `truffle test`



## Truffle migrate

### Transaction






## JSON config file

```
  "networks": {
    "5777": {
      "events": {},
      "links": {},
      "address": "0x948231d3CeD48A6FCf0A925144Ca9be47a270dd9",
      "transactionHash": "0
     x60829cdcf064ad5e5efc896b87e308d5fef31ba0c42450ec396d888b2a30dea9"
    }
```

Figure: JSON config file for truffle project, `myFirstContract.json`



## Unit Testing

- test the functions and their results
- expected results and test for failures
- compare expected and actual for match



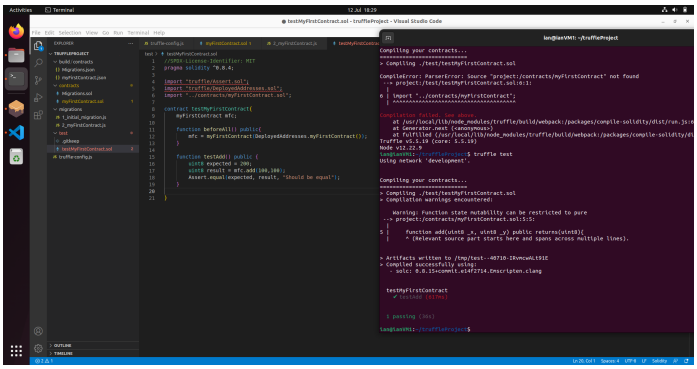
## Truffle Test

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;
import "truffle/Assert.sol";
import "truffle/DeployedAddresses.sol";
import "../contracts/myFirstContract.sol";
contract testMyFirstContract{
    myFirstContract mfc;
    function beforeAll() public{
        mfc = myFirstContract(DeployedAddresses.myFirstContract());
    }
    function testAdd() public {
        uint8 expected = 200;
        uint8 result = mfc.add(100,100);
        Assert.equal(expected, result, "Should be equal");
    }
}
```

Figure: Truffle test file for truffle project, `test/testMyFirstContract.sol`



## Truffle Test Output



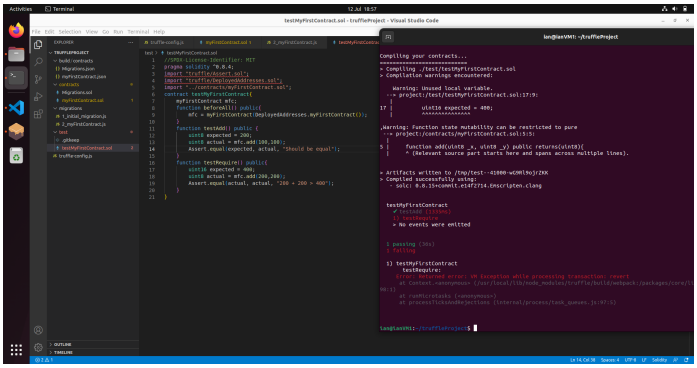
## Truffle Test

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;
import "truffle/Assert.sol";
import "truffle/DeployedAddresses.sol";
import "../contracts/myFirstContract.sol";
contract testMyFirstContract{
    myFirstContract mfc;
    function beforeAll() public{
        mfc = myFirstContract(DeployedAddresses.myFirstContract());
    }
    function testAdd() public {
        uint8 expected = 200;
        uint8 actual = mfc.add(100,100);
        Assert.equal(expected, actual, "Should be equal");
    }
    function testRequire() public{
        uint16 expected = 400;
        uint8 actual = mfc.add(200,200);
        Assert.equal(actual, actual, "200 + 200 > 400");
    }
}
```

Figure: Truffle test file for truffle project, `test/testMyFirstContract1.sol`

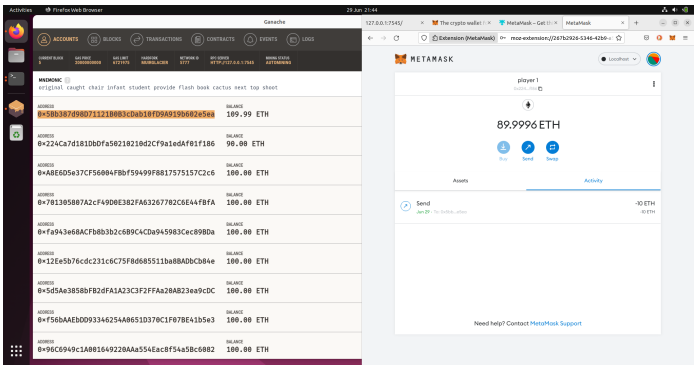


## Truffle Test Output



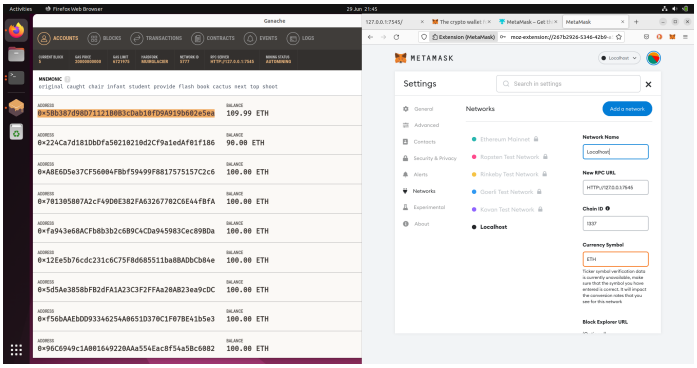


## Ganache Network
with MetaMask



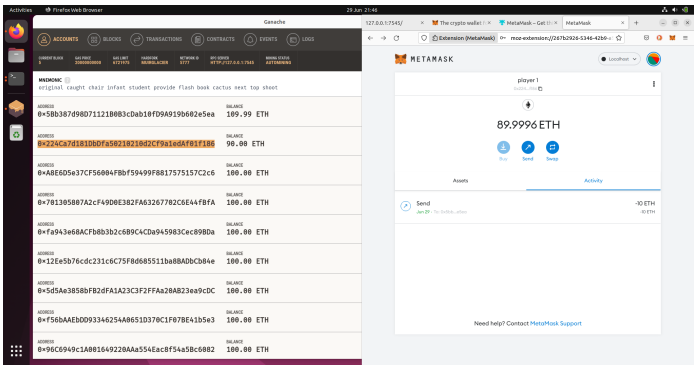


## Add Account to MetaMask



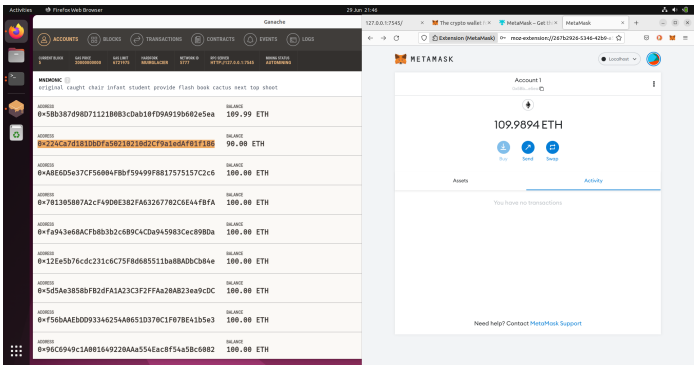


## MetaMask Account Added

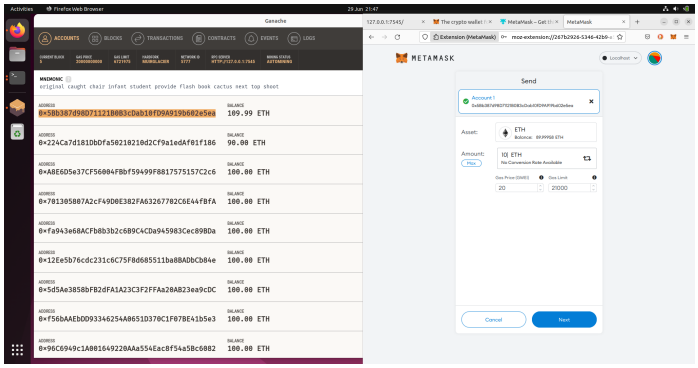




## Another MetaMask Account Added

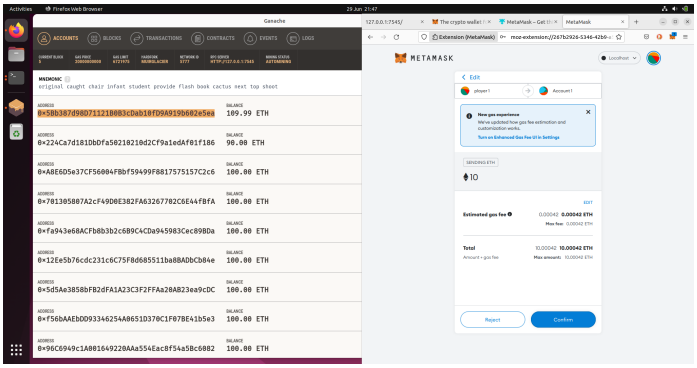


## Send Ether between Accounts



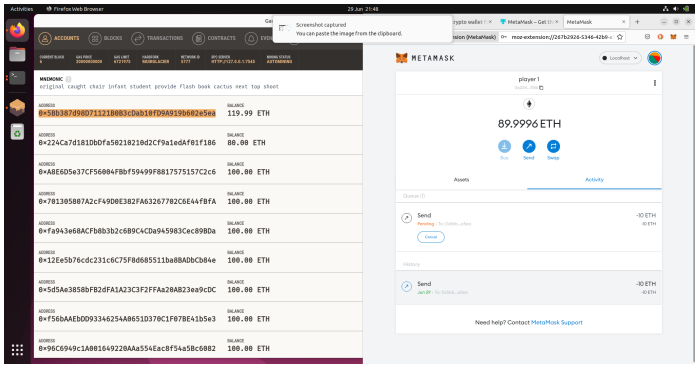


## Confirm Ether sent between Accounts





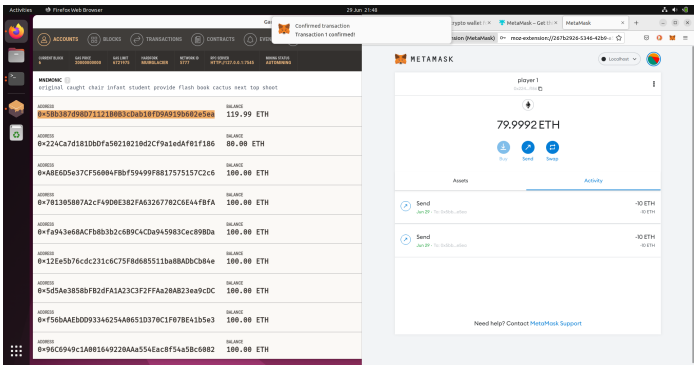
## Pending TX





## TX confirmed

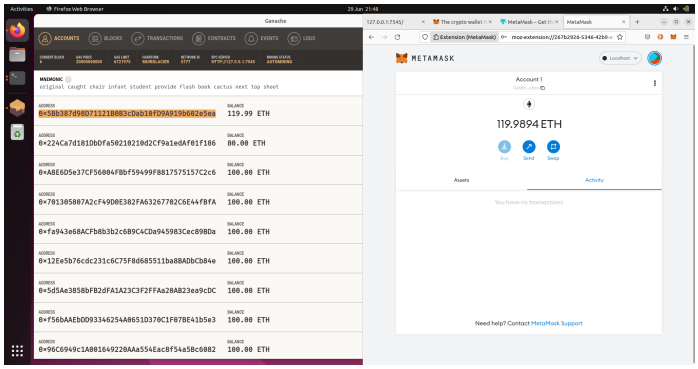
player 1





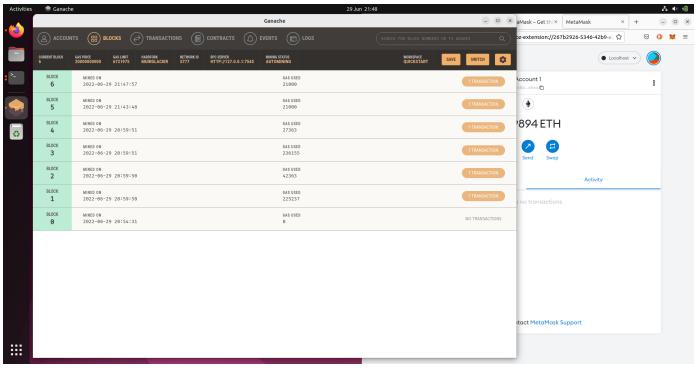
## TX confirmed

Account 1

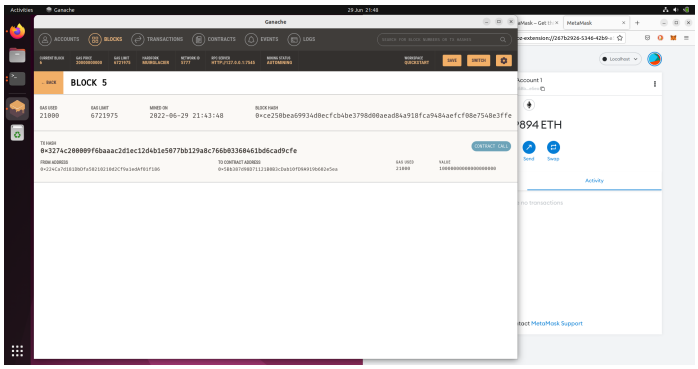




## Blocks

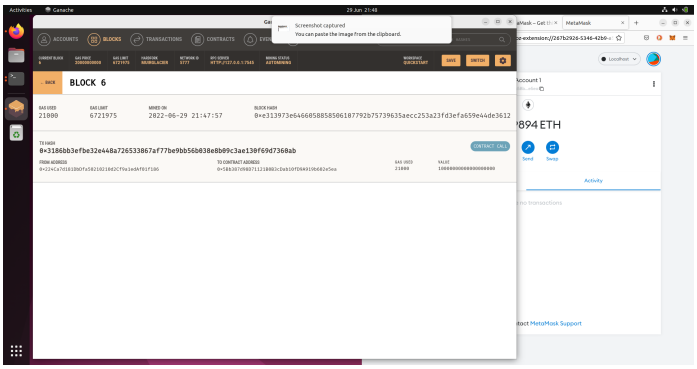


## An individual block
Block 5






## An individual block
Block 6






## Network Summary

- Ganache
- MetaMask Wallets
- Make TX
- Examine TX
- Network



## Summary

- Network
- Accounts
- Exceptions
- Unit Testing
- Modifiers



## Reading

- Chaper 8 in [1]



## References I

[1] Ritesh Modi. *Solidity Programming Essentials*. Packt, 2018. ISBN: 978-1-78883-138-3.