

## SMART CONTRACT PROGRAMMING

### INITIAL SETUP

```

-> mkdir lab4
-> cd lab4/
lab4> sudo add-apt-repository ppa:ethereum/ethereum
[sudo] password for user1:

More info: https://launchpad.net/~ethereum/+archive/ubuntu/ethereum
Press [ENTER] to continue or ctrl-c to cancel adding it

gpg: keyring `/tmp/tmpq5skw6n/_secring.gpg' created
gpg: keyring `/tmp/tmpq5skw6n/_pubring.gpg' created
gpg: requesting key 923F6CA9 from hkp server keyserver.ubuntu.com
gpg: /tmp/tmpq5skw6n/_trustdb.gpg: trustdb created
gpg: key 923F6CA9: public key "Launchpad PPA for Ethereum" imported
gpg: Total number processed: 1
gpg:          imported: 1  (RSA: 1)
OK
lab4> sudo apt-get update
Hit:1 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease
Hit:4 http://ppa.launchpad.net/ethereum/ethereum/ubuntu xenial InRelease
Hit:5 http://security.ubuntu.com/ubuntu xenial-security InRelease
Hit:6 https://deb.nodesource.com/node_8.x xenial InRelease
Reading package lists... Done
lab4> sudo apt-get install solc
Reading package lists... Done
Building dependency tree
Reading state information... Done
solc is already the newest version (1:0.4.25-0ubuntu1-xenial).
The following packages were automatically installed and are no longer required:
  gyp javascript-common libjs-inherits libjs-jquery libjs-node-uuid
  libjs-underscore libssl-dev libssl-doc libuv1 libuv1-dev node-abbrev
  node-ansi node-ansi-color-table node-archy node-async node-block-stream
  node-combined-stream node-cookie-jar node-delayed-stream node-forever-agent
  node-form-data node-fstream node-fstream-ignore node-github-url-from-git
  node-glob node-graceful-fs node-gyp node-inherits node-init
  node-json-stringify-safe node-lockfile node-lru-cache node-mime
  node-minimatch node-mkdirp node-mute-stream node-node-uuid node-nopt
  node-normalize-package-data node-npmlog node-once node-osenv node-qs
  node-read node-read-package-json node-request node-retry node-rimraf
  node-semver node-sha node-sigmund node-slide node-tar node-tunnel-agent
  node-underscore node-which python-pkg-resources zlib1g-dev
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 188 not upgraded.

```

M  
R  
U  
D  
H  
U  
L  
A

We install Solc and set up the Ethereum client for this lab. We use the following commands for Solc installation:

```

sudo add-apt-repository ppa:Ethereum/Ethereum
sudo apt-get update
sudo apt-get install solc

```

## EXERCISE 1a: Hello-world Contract with Remix

```

Terminal
lab4> cd ~
-> sudo cp -r lab3 lab4
-> cd lab4
lab4> sudo gedit hello.sol

(gedit:30637): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was not provided by any .service files

** (gedit:30637): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-spell-enabled not supported
** (gedit:30637): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-encoding not supported
** (gedit:30637): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
lab4> solc -o . --bin --abi hello.sol
lab4> ls
hello.abi  hello.bin  hello.sol  lab3

```

We create a smart contract file and copy the text of hello world program provided to us and save it with an extension of '.sol', then we compile the code. Through the compilation we will be generating two files one 'hello.abi' (it is an interface through which the compiled codes interact with the execution platform) and another file is 'hello.bin' this file stores the binary code.

```

Terminal
lab4> cat hello.abi
[{"constant":true,"inputs":[],"name":"greet","outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"function"}, {"constant":false,"inputs":[{"name":"_greeting","type":"string"}],"name":"greeter","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"}]lab4>
lab4> cat hello.bin
608060405234801561001057600080fd5b506102d7806100206000396000f30060806040526004
361061004c576000357c010000000000000000000000000000000000000000000000000000000000000000
900463fffffffff168063cfae321714610051578063faf27bca146100e1575b600080fd5b348015
61005d57600080fd5b5061006661014a565b604051808060200182810382528381815181526020
0191508051906020019080838360005b838110156100a657808201518184015260208101905061
008b565b505050905090810190601f1680156100d35780820380516001836020036101000a03
1916815260200191505b509250505060405180910390f35b3480156100ed57600080fd5b506101
48600480360381019080803590602001908201803590602001908080601f016020809104026020
0160405190810160405280939291908181526020018383808284378201915050505050919291
929050506101ec565b005b606060008054600181600116156101000203166002900480601f01
60208091042602001604051908101604052809291908181526020018280546001816001161561
01000203166002900480156101e25780601f106101b75761010080835404028352916020019161
01e2565b820191906000526020600020905b8154815290600101906020018083116101c5578290
03601f168201915b50505050905090565b806000908051906020019061020292919061020656
5b5050565b828054600181600116156101000203166002900490600052602060002090601f0160
20900481019282601f1061024757805160ff1916838001178555610275565b8280016001018555
8215610275579182015b82811115610274578251825591602001919060010190610259565b5b50
90506102829190610286565b5090565b6102a891905b808211156102a457600081600090555060
010161028c565b5090565b905600a165627a7a72305820ce00381fb6dde4646ec8e969455fd5a3
9a0ce80ce80b5dde4fdda68e3122ee400029lab4>
```

The above screenshot displays the original contents of the two files generated after compilation.

To deploy the compiled files, we modify the contents of both the generated files as shown in the above screenshot.

```
Terminal
-> cd lab3
lab3> geth --datadir bkc_data init ~/lab3/genesis.json
INFO [10-14|18:11:31] Maximum peer count
INFO [10-14|18:11:31] Allocated cache and file handles
bkc_data/geth/chaindata cache=16 handles=16
INFO [10-14|18:11:33] Persisted trie from memory database
038μs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [10-14|18:11:33] Successfully wrote genesis state
hash=b913d0...07d3df
INFO [10-14|18:11:33] Allocated cache and file handles
bkc_data/geth/lightchaindata cache=16 handles=16
INFO [10-14|18:11:33] Persisted trie from memory database
132μs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [10-14|18:11:33] Successfully wrote genesis state
hash=b913d0...07d3df
lab3> geth --datadir bkc_data --networkid 89992018 --bootnodes enode://d3cd4e70fe7ad1dd7f
b23539c53982e42816b4218cc370e8af13945f7b5e2b4a288f8b949dbdba6a998c9141266a0df61523de74490
c91fc1e3d538b299bb8ab@128.230.208.73:30301 console 2>console.log
Fatal: Error starting protocol stack: listen udp :30303: bind: address already in use
lab3> geth --datadir bkc_data --networkid 89992018 --bootnodes enode://d3cd4e70fe7ad1dd7f
b23539c53982e42816b4218cc370e8af13945f7b5e2b4a288f8b949dbdba6a998c9141266a0df61523de74490
c91fc1e3d538b299bb8ab@128.230.208.73:30301 console 2>console.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.9-stable-ff9b1461/linux-amd64/go1.10
coinbase: 0x1a0aeb2893f19e854eabdc4822b51693cd1321c5
at block: 90359 (Sat, 06 Oct 2018 18:28:57 EDT)
datadir: /home/user1/lab3/bkc_data
modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 w
eb3:1.0

> admin.addPeer("enode://d2547d500b1e982ac93a6ce1dbf34cff6545987740313373ccecb28e095c6ce4
294e5cf4be2f002672d30fb717b8bd05e1a12163b24743b907bb7d2c37415928@[128.230.208.73]:30303")
true
```

We deploy the smart contract by connecting to the Ethereum network as shown above.



```

Terminal
> loadScript("hello.bin")
Unlock account 0x1a0aeb2893f19e854eabdc4822b51693cd1321c5
Passphrase:
true
> helloVar.greeter.sendTransaction("Hello",{from:eth.accounts[0],gas:700000})
TypeError: Cannot access member 'sendTransaction' of undefined
    at <anonymous>:1:1

> helloVar.greeter.sendTransaction("Hello",{from:eth.accounts[0],gas:700000})
TypeError: Cannot access member 'sendTransaction' of undefined
    at <anonymous>:1:1

> helloVar.greeter.sendTransaction("Hello",{from:eth.accounts[0],gas:700000}, "mrudhula")
TypeError: Cannot access member 'sendTransaction' of undefined
    at <anonymous>:1:1

> helloVar.greeter.sendTransaction("Hello", "mrudhula", {from:eth.accounts[0],gas:700000})
TypeError: Cannot access member 'sendTransaction' of undefined
    at <anonymous>:1:1

> helloVar.greeter.personal.sendTransaction("Hello", {from:eth.accounts[0],gas:700000}, "mrudhula")
TypeError: Cannot access member 'personal' of undefined
    at <anonymous>:1:1

> helloVar.greeter.sendTransaction("Hello", {from:eth.accounts[0],gas:700000})
TypeError: Cannot access member 'sendTransaction' of undefined
    at <anonymous>:1:1

> miner.start[10]
undefined
> miner.start(10)
null

```

Then we load and run the script to deploy the smart contract. When we load and run ‘hello.abi’ it gives true and when we load and run ‘hello.bin’ it unlocks our account and asks for our passphrase. Then when we check the console log file we find a contract address and transaction hash value as shown below. Now we do send transaction giving the contract name and the initial function ‘greeter’ name through with we pass the arguments. As I tried multiple times I notice that I do not get any results. Then we start mining to make the process faster and start the send transaction again, we keep trying till we achieve a transaction hash value. If suppose we do not achieve in initially try (In my case) I reloaded and re-ran the script to deploy the smart contract after I started mining and then when I tried the send transaction it was successful, and I got a transaction hash value back. As shown below the screenshot of console log file .



```

console.log (~/lab3) - gedit
Open Save
mgas=0.000 elapsed=123.080ms mgasps=0.000 number=121463 hash=5c8ba6...75bb94
cache=224.07kB
INFO [10-14|18:13:17] Imported new chain segment blocks=192 txs=0
mgas=0.000 elapsed=101.925ms mgasps=0.000 number=121655 hash=417c95...b99b3d
cache=224.07kB
INFO [10-14|18:13:17] Imported new chain segment blocks=192 txs=0
mgas=0.000 elapsed=122.805ms mgasps=0.000 number=121847 hash=e0b8e0...d702fa
cache=224.07kB
INFO [10-14|18:13:17] Imported new chain segment blocks=192 txs=0
mgas=0.000 elapsed=144.690ms mgasps=0.000 number=122039 hash=2c78e6...3a30d9
cache=224.07kB
INFO [10-14|18:13:18] Imported new chain segment blocks=192 txs=0
mgas=0.000 elapsed=103.473ms mgasps=0.000 number=122231 hash=3bc3a1...0f4d9e
INFO [10-14|18:13:18] Submitted contract creation
fullhash=0x05d744f91270a8bcb40c2359846cc1833a64def5c174b46506dbfdfa7b3dc3f6
contract=0x26D05fB40b58Ab59D5c818afa9Ba6c3188fe004B
INFO [10-14|18:13:18] Imported new chain segment blocks=192 txs=0

```

A screenshot of a Mac OS X terminal window titled "Terminal". The terminal is displaying Ethereum command-line interface (CLI) commands. The user is loading a script ("hello.abi"), unlocking an account, sending transactions, and calling the "greet" function of the contract. The output shows the transaction hash and the response "Hello".

```

> loadScript("hello.abi")
true
> loadScript("hello.bin")
Unlock account 0x1a0aeb2893f19e854eabdc4822b51693cd1321c5
Passphrase:
true
> helloVar.greeter.sendTransaction("Hello", {from:eth.accounts[0],gas:700000})
"0x34886fcf9c9d80148534f3edb444cebf124ef2138c058d711280a6dd988c5065"
> helloVar.greet.call()
""

> helloVar.greeter.sendTransaction("Hello", {from:eth.accounts[0],gas:700000})
"0x0fb488d75038e2477882f23dd3044d19b8bd95aabe2d6700b5495b25e2d26278"
> helloVar.greet.call()
"Hello"
>

```

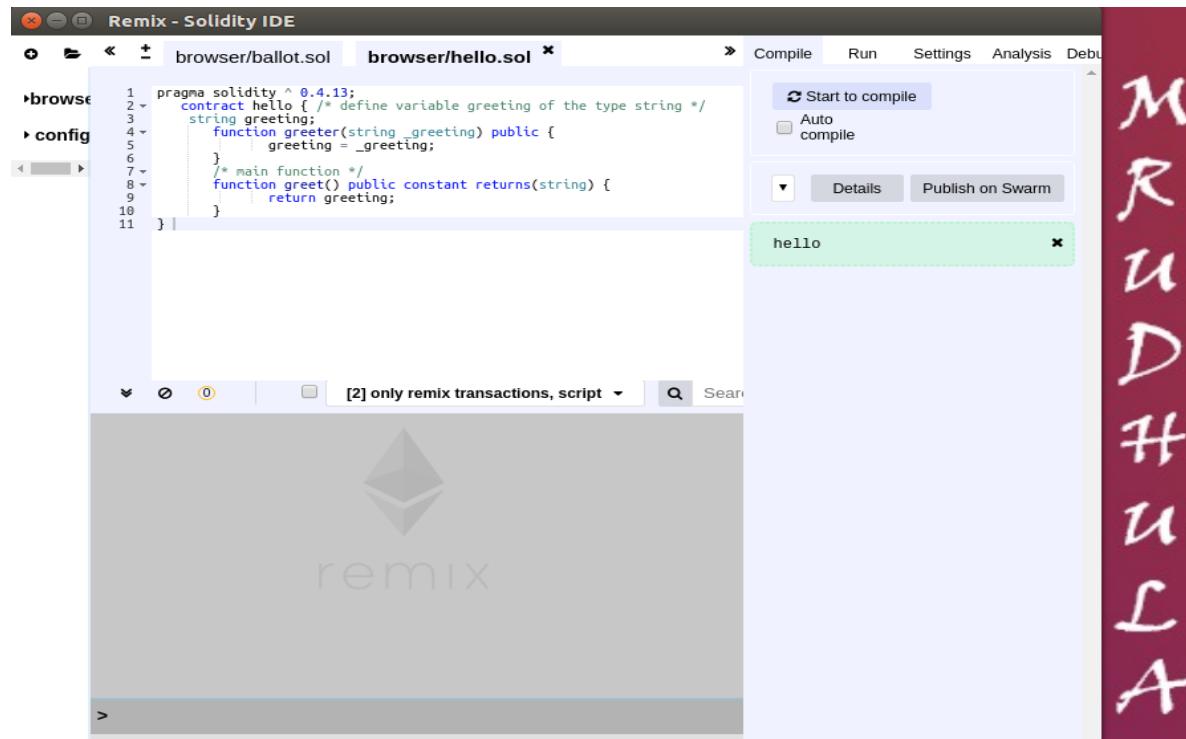
We execute the greet function in which we get the hello displayed. We check our console log file ones again to see the new contract address and the transaction hash value after executing the functions sequentially.

```

INFO [10-14|19:32:50] Commit new mining work                               number=139501 txs=0
uncles=0 elapsed=133.089µs
INFO [10-14|19:33:04] Submitted contract creation
fullhash=0x0fc4a34465e8556b35b75c11236714ec6a2b0872dcab6526dbd8a6a09586a039
contract=0xc3026d8884334260DA0fDF3A57B9698f66301b17
INFO [10-14|19:33:07] Imported new chain segment      blocks=1    txs=0
mgas=0.000 elapsed=217.405ms mgasps=0.000   number=139501 hash=53b697..676dec
cache=270.98kB
INFO [10-14|19:33:07] Commit new mining work                               number=139502 txs=1
uncles=0 elapsed=361.691µs
INFO [10-14|19:33:08] Imported new chain segment      blocks=1    txs=1
mgas=0.246 elapsed=85.051ms mgasps=2.894   number=139502 hash=5e0eed...1fae3b
cache=272.59kB
INFO [10-14|19:33:08] Commit new mining work                               number=139503 txs=0
uncles=0 elapsed=145.975µs
INFO [10-14|19:33:13] Imported new chain segment      blocks=1    txs=0
mgas=0.000 elapsed=165.933ms mgasps=0.000   number=139503 hash=0ef7c3..02458c
cache=272.59kB
INFO [10-14|19:33:13] Commit new mining work                               number=139504 txs=0
uncles=0 elapsed=118.877µs
INFO [10-14|19:33:13] ⚡ block reached canonical chain      number=139498 hash=c70c98...
83c572
INFO [10-14|19:33:18] Submitted transaction
fullhash=0x34886fcf9c9d80148534f3edb444cebf124ef2138c058d711280a6dd988c5065
recipient=0xc3026d8884334260DA0fDF3A57B9698f66301b17

```

## EXERCISE 1b: Hello-world Contract with Solc and On-Campus Ethereum



The screenshot shows the Remix Solidity IDE interface. In the top navigation bar, the tabs 'browser/ballot.sol' and 'browser/hello.sol' are selected. The main code editor contains the following Solidity code:

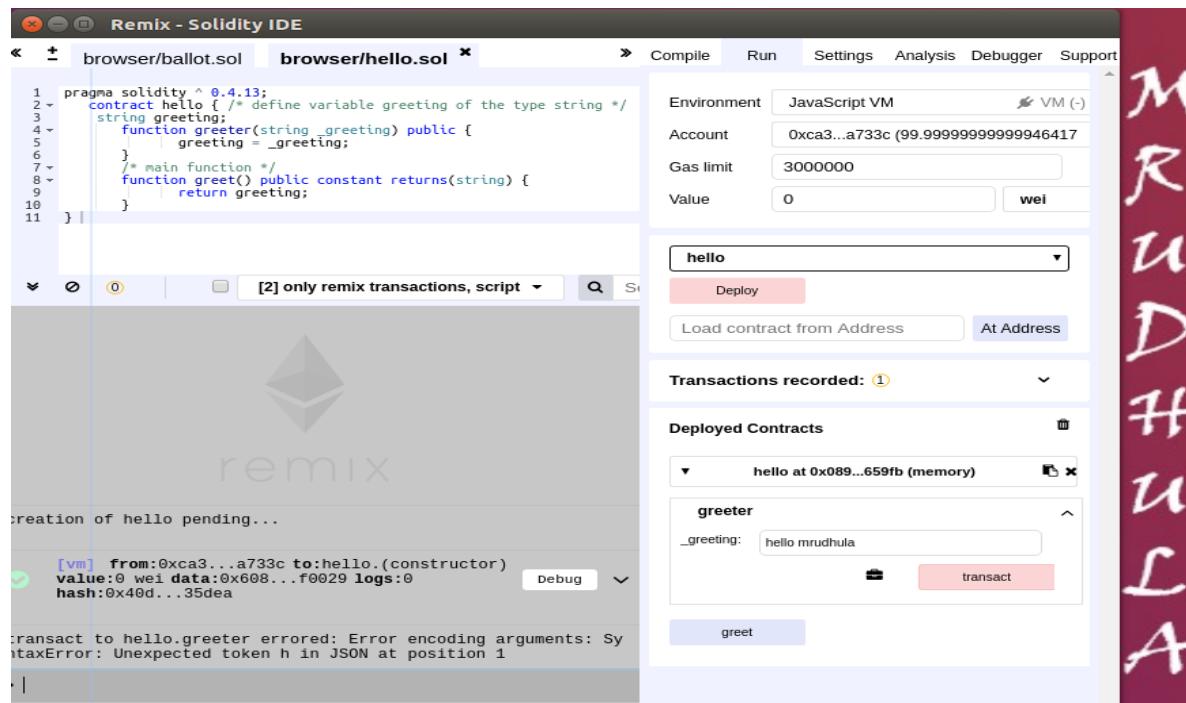
```

1 pragma solidity ^ 0.4.13;
2 contract hello { /* define variable greeting of the type string */
3     string greeting;
4     function greeter(string _greeting) public {
5         greeting = _greeting;
6     }
7     /* main function */
8     function greet() public constant returns(string) {
9         return greeting;
10    }
11 }

```

To the right of the code editor, there is a sidebar with buttons for 'Start to compile' (which is highlighted in blue), 'Auto compile', 'Details', and 'Publish on Swarm'. A green box labeled 'hello' is displayed below these buttons. At the bottom of the interface, there is a search bar and a transaction history section showing '[2] only remix transactions, script'.

We run the given hello-world program on a command-line programming framework solc. We first compile the program, when program is successfully compiled it will return the program name.



The screenshot shows the Remix Solidity IDE interface after compilation. The code editor remains the same. The sidebar now includes additional options: 'Environment' set to 'JavaScript VM', 'Account' set to '0xca3...a733c (99.99999999999946417)', 'Gas limit' set to '3000000', and 'Value' set to '0 wei'. Below these settings, a dropdown menu is set to 'hello'. A pink button labeled 'Deploy' is visible. The 'Transactions recorded:' section shows a single entry: 'creation of hello pending...'. The bottom status bar displays a transaction log: '[vm] from:0xca3...a733c to:hello.(constructor) value:0 wei data:0x608...f0029 logs:0 hash:0x40d...35dea'. A 'Debug' button is also present in the status bar.

Then we run the program after compiling it successfully. We get an option bar deploy and we click on that to deploy the contract. We give a value for the first function greeter. And click on transact.

```

1 pragma solidity ^0.4.13;
2 contract hello { /* define variable greeting of the type string */
3     string greeting;
4     function greeter(string _greeting) public {
5         greeting = _greeting;
6     }
7     /* main function */
8     function greet() public constant returns(string) {
9         return greeting;
10    }
11 }
```

**Environment:** JavaScript VM, VM (-)  
**Account:** 0xca3...a733c (99.9999999999942057)  
**Gas limit:** 3000000  
**Value:** 0 wei

**Deploy**

**Transactions recorded:** ②

**Deployed Contracts:**

- hello at 0x089...659fb (memory)

**greeter**

\_greeting: string

transact

greet

status	0x1 Transaction mined and execution succeeded
transaction hash	0xe69b504c44875e24d593e46aa20d33dc956daaa303957a796bfc7062594eae
from	0xca35b7d915458ef540ade0068dfe2f44e8fa733c
to	hello.greeter(string) 0x08970fed061e7747cd9a38d680a601510cb659fb
gas	3000000 gas
transaction cost	43609 gas
execution cost	20929 gas
hash	0xe69b504c44875e24d593e46aa20d33dc956daaa303957a796bfc7062594eae
input	0xfaf...00000
decoded input	{ "string _greeting": "hello mrudhula" }
decoded output	{}
logs	[]
value	0 wei

After we transact, we can see the transaction on the output screen and it shows the decoded input.

The screenshot shows the Remix - Solidity IDE interface. On the left, there are two tabs: "browser/ballot.sol" and "browser/hello.sol". The "browser/hello.sol" tab is active, displaying the following Solidity code:

```

1 pragma solidity ^0.4.13;
2 contract hello { /* define variable greeting of the type string */
3     string greeting;
4     function greeter(string _greeting) public {
5         greeting = _greeting;
6     }
7     /* main function */
8     function greet() public constant returns(string) {
9         return greeting;
10    }
11 }

```

Below the code editor, a transaction details panel is visible, showing a call to the "greet" function of the "hello" contract. The transaction hash is 0x1924273ebbe078fb1b033630262241ffc67fe173, from address 0xca35b7d915458ef540ade6068dfe2f44e8fa733c, to address 0x08970fed061e7747cd9a38d680a601510cb659fb, with a gas cost of 22608 and an execution cost of 1426.

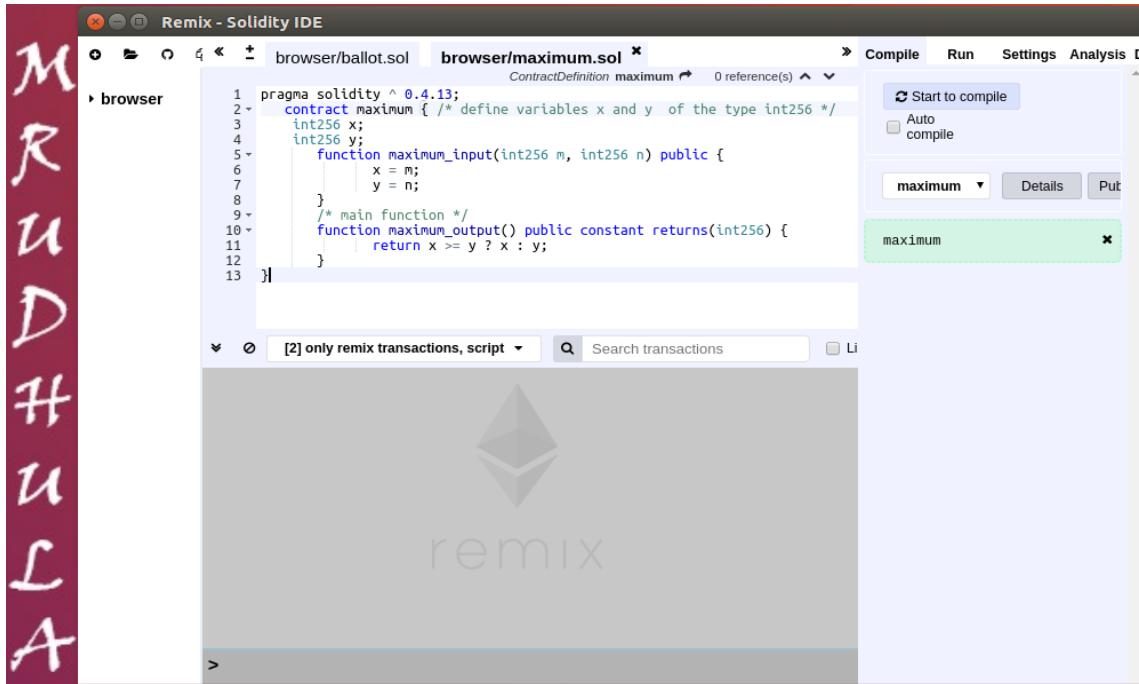
On the right side, the deployment interface is shown. It includes fields for Environment (JavaScript VM), Account (0xca3...a733c), Gas limit (3000000), and Value (0 wei). A dropdown menu shows the deployed contract "hello" at address 0x089...659fb (memory). Under the "greeter" section, the "greeting" field is set to "string". A "transact" button is available to interact with the contract. The output of the "greet" function is displayed as "0: string: hello mrudhula".

After that we click on the greet function option which will provide us the decoded output. We can check this on the output screen as shown above which will print the string that we provided to the greeter function.

M  
R  
U  
D  
H  
U  
L  
A

## EXERCISE 2: Find the Maximum

In Solc



The screenshot shows the Remix Solidity IDE interface. On the left, there is a vertical sidebar with the text "MRUDHULAA". The main workspace contains two tabs: "browser/ballot.sol" and "browser/maximum.sol". The "maximum.sol" tab displays the following Solidity code:

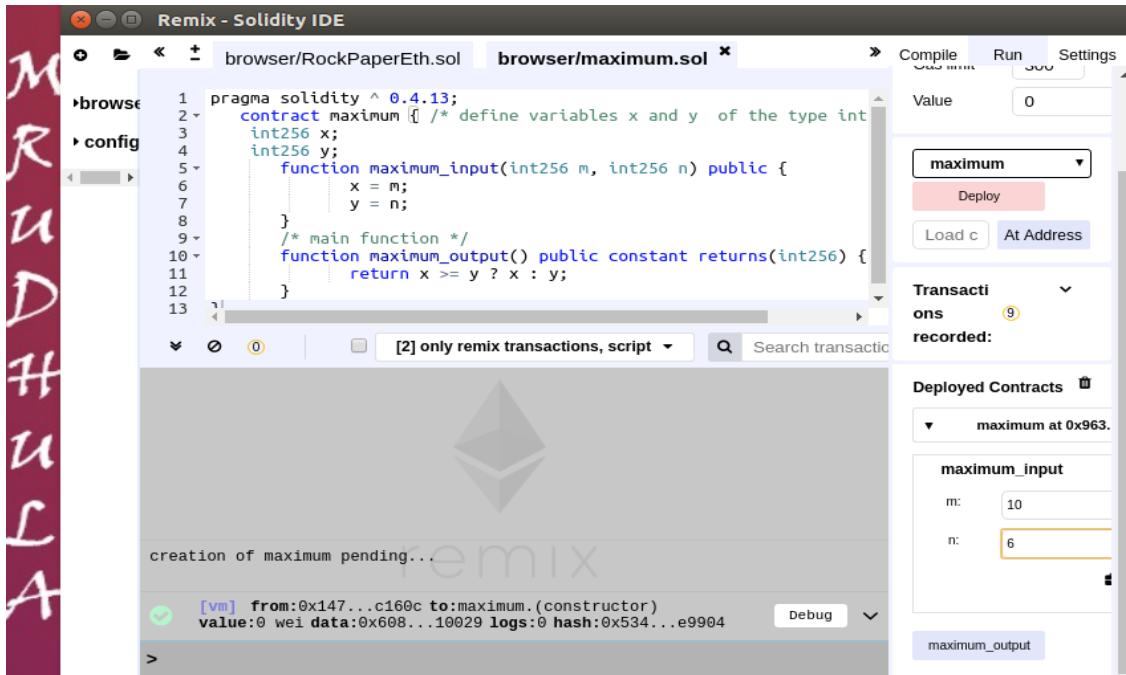
```

1 pragma solidity ^ 0.4.13;
2 contract maximum { /* define variables x and y of the type int256 */
3     int256 x;
4     int256 y;
5     function maximum_input(int256 m, int256 n) public {
6         x = m;
7         y = n;
8     }
9     /* main function */
10    function maximum_output() public constant returns(int256) {
11        if (x >= y) return x; else return y;
12    }
13 }

```

The right side of the interface includes a "Compile" button, a "Start to compile" checkbox, and a dropdown menu showing "maximum". A green box highlights the word "maximum" in the dropdown.

We write program for finding the maximum numbers and we take an input of x and y. The two functions here are maximum\_input and maximum\_output. In maximum\_output function we give a condition to check both the values and print out the maximum value. We compile the program. We see the program is successfully compiled as it returns the program name in green.



The screenshot shows the Remix Solidity IDE interface with the "maximum.sol" tab selected. The code is identical to the one in the previous screenshot. On the right, the "Run" tab is active, showing deployment options like "Deploy" and "At Address". Below that, the "Transactions recorded" section shows a pending transaction: "creation of maximum pending...". The "Deployed Contracts" section shows the contract "maximum at 0x963." with its methods "maximum\_input" and "maximum\_output". The "maximum\_input" method has inputs "m: 10" and "n: 6". At the bottom, the transaction details show "[vm] from:0x147...c160c to:maximum.(constructor) value:0 wei data:0x608...10029 logs:0 hash:0x534...e9904".

We then deploy the contract we give details for the first function that is maximum\_input which takes two values which are of integer type.

```

1 pragma solidity ^ 0.4.13;
2 contract maximum { /* define variables x and y of the type int256 */
3     int256 x;
4     int256 y;
5     function maximum_input(int256 m, int256 n) public {
6         x = m;
7         y = n;
8     } /* main function */
9     function maximum_output() public constant returns(int256) {
10        return x >= y ? x : y;
11    }
12 }
```

[vm] from:0x147...c160c to:maximum.maximum\_input(int256,int256) 0x963...eb069 value:0 wei data:0x5f1...00006 logs:0 hash:0xca5...b6c14

status	0x1 Transaction mined and execution succeed
transaction hash	0xca551856492849d061c800056c9fe1be6b7faf2fe525534f3
from	0x14723a09acff6d2a60dcdf7aa4aff308fddc160c
to	maximum.maximum_input(int256,int256) 0x9635e132729aa83b126ab8b159194196b5eeb069
gas	3000000 gas
transaction cost	61904 gas
execution cost	40248 gas
hash	0xca551856492849d061c800056c9fe1be6b7faf2fe525534f3
input	0x5f1...00006
decoded input	{ "int256 m": "10", "int256 n": "6" }
decoded output	{}
logs	[]
value	0 wei

Deployed Contracts

- maximum
 

m:	int256
n:	int256
- maximum\_output

We transact the function and check the console log and we see in the decoded input are the two value that we had given to find the maximum value.

```

1 pragma solidity ^ 0.4.13;
2 contract maximum { /* define variables x and y of the type int256 */
3     int256 x;
4     int256 y;
5     function maximum_input(int256 m, int256 n) public {
6         x = m;
7         y = n;
8     } /* main function */
9     function maximum_output() public constant returns(int256) {
10        return x >= y ? x : y;
11    }
12 }
```

call to maximum.maximum\_output

[CALL] from:0x14723a09acff6d2a60dcdf7aa4aff308fddc160c to:maximum.maximum\_output() data:0xff6...791f4

transaction hash	0x6f898538041e2e3e7f397e791f4cca06423d999e03657fa3a4
from	0x14723a09acff6d2a60dcdf7aa4aff308fddc160c
to	maximum.maximum_output() 0x9635e132729aa83b126ab8b159194196b5eeb069
transaction cost	22123 gas (Cost only applies when called by a contract)
execution cost	851 gas (Cost only applies when called by a contract)
hash	0x6f898538041e2e3e7f397e791f4cca06423d999e03657fa3a4
input	0xff6...791f4
decoded input	{}
decoded output	{ "0": "int256: 10" }
logs	[]

Deployed Contracts

- maximum
 

m:	int256
n:	int256
- maximum\_output
 

0: int256: 10
---------------

We now click on the next function that is the maximum\_output function we see that we have obtained the maximum value out of the two integer numbers we had passed. We check the console log to see if our answer is recorded and visible at the decoded output as shown above.

## On Terminal

We write the same program that we did in solc. And repeat the steps done for exercise 1a.

```
x - Terminal
~> cd lab3/
lab3> sudo gedit maximum.sol
[sudo] password for user1:
** (gedit:4172): WARNING **: Set document metadata failed: Setting attribute met
adata::gedit-position not supported
lab3> solc -o . --bin --abi maximum.sol
lab3> ls
bkc_data      genesis.json  hello.bin   maximum.abi   maximum.sol
console.log   hello.abi     hello.sol    maximum.bin
lab3> ls | grep maximum.
maximum.abi
maximum.bin
maximum.sol
lab3> 
```

```

Terminal
~> cd lab3/
lab3> cat maximum.abi
var maximumContract = eth.contract([{"constant":false,"inputs":[{"name":"m","type":"int256"}, {"name":"n","type":"int256"}],"name":"maximum_input","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"}, {"constant":true,"inputs":[],"name":"maximum_output","outputs":[{"name":"","type":"int256"}],"payable":false,"stateMutability":"view","type":"function"}])
lab3> cat maximum.bin
personal.unlockAccount(eth.accounts[0])
var maximumVar = maximumContract.new({from:eth.accounts[0],
data:"0x608060405234801561001057600080fd5b50610105806100206000396000f30060806040
52600436106049576000357c01000000000000000000000000000000000000000000000000000000000000
00900463ffffffff1680635f16852014604e578063ff6791f4146082575b600080fd5b3480156059
57600080fd5b50608060048036038101908080359060200190929190803590602001909291905050
5060aa565b005b348015608d57600080fd5b50609460bc565b604051808281526020019150506040
5180910390f35b81600081905550806001819055505050565b6000600154600054121560d0576001
5460d4565b6000545b9050905600a165627a7a723058208591e8c5e4e22d38985b8d8c12b2b67790
689f3b972c69e7b9163e7f77bd25200029",gas:500000})
lab3>

```

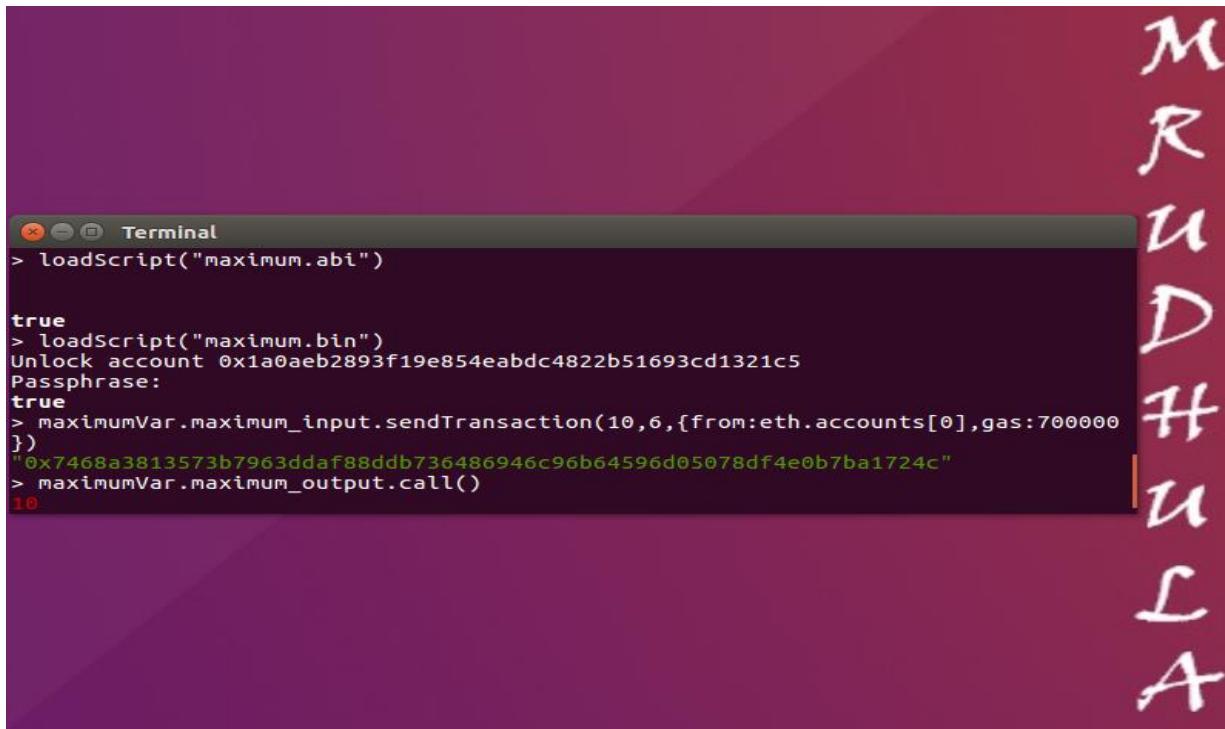
```

Terminal
lab3> geth --datadir bkc_data init ~/lab3/genesis.json
INFO [10-15|22:33:22] Maximum peer count          ETH=25 LES=0 total=25
INFO [10-15|22:33:22] Allocated cache and file handles   database=/home/usr1/lab3/bkc_data/geth/chaindata
INFO [10-15|22:33:24] Persisted trie from memory database   nodes=0 size=0.00
B time=1.935µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [10-15|22:33:24] Successfully wrote genesis state   database=chaindata
hash=b913d0...07d3df
INFO [10-15|22:33:24] Allocated cache and file handles   database=/home/usr1/lab3/bkc_data/geth/lightchaindata
INFO [10-15|22:33:24] Persisted trie from memory database   nodes=0 size=0.00
B time=2.053µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [10-15|22:33:24] Successfully wrote genesis state   database=lightchaindata
hash=b913d0...07d3df
lab3> geth --datadir bkc_data --networkid 89992018 --bootnodes enode://d3cd4e70fe7ad1dd7fb2359c53982e42816b4218cc370e8af13945f7b5e2b4a288f8b949dbba6a998c9141266a0df61523de74490c91fc1e3d538b299bb8ab@128.230.208.73:30301 console 2>console.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.9-stable-ff9b1461/linux-amd64/go1.10
coinbase: 0x1a0aeb2893f19e854eabdc4822b51693cd1321c5
at block: 90359 (Sat, 06 Oct 2018 18:28:57 EDT)
datadir: /home/usr1/lab3/bkc_data
modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

> admin.addPeer("enode://d2547d500b1e982ac93a6ce1dbf34cff6545987740313373ccecb28e095c6ce4294e5cf4be2f002672d30fb717b8bd05e1a12163b24743b907bb7d2c37415928@[128.230.208.73]:30303")
true

```



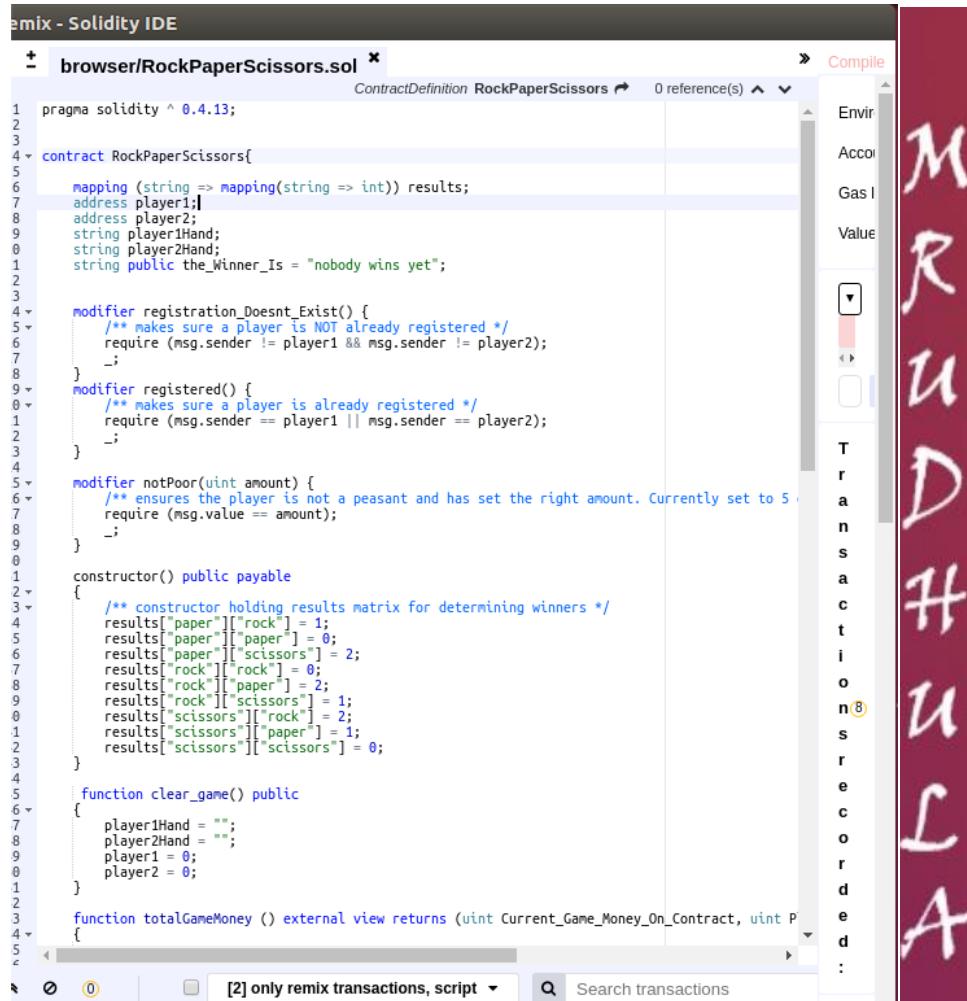
```
Terminal
> loadScript("maximum.abi")

true
> loadScript("maximum.bin")
Unlock account 0x1a0aeb2893f19e854eabdc4822b51693cd1321c5
Passphrase:
true
> maximumVar.maximum_input.sendTransaction(10,6,{from:eth.accounts[0],gas:700000
})
"0x7468a3813573b7963ddaf88ddb736486946c96b64596d05078df4e0b7ba1724c"
> maximumVar.maximum_output.call()
10
```

But here in send transaction we 4 parameters whereas in hello-world program we had given three. Here we give two integer numbers that are to be compared to find maximum value, the account detail from and gas value.

## EXERCISE 2: Rock-paper-scissors game

Below is the code for the program rock paper scissors game.



The screenshot shows the Solidity IDE interface with the file "browser/RockPaperScissors.sol" open. The code defines a contract named "RockPaperScissors" with various functions and modifiers. A vertical watermark "MRUDHULA" is visible on the right side of the screen.

```
pragma solidity ^0.4.13;

contract RockPaperScissors{
    mapping (string => mapping(string => int)) results;
    address player1;
    address player2;
    string player1Hand;
    string player2Hand;
    string public the_Winner_Is = "nobody wins yet";

    modifier registration_Doesnt_Exist() {
        /** makes sure a player is NOT already registered */
        require (msg.sender != player1 && msg.sender != player2);
        _;
    }

    modifier registered() {
        /** makes sure a player is already registered */
        require (msg.sender == player1 || msg.sender == player2);
        _;
    }

    modifier notPoor(uint amount) {
        /** ensures the player is not a peasant and has set the right amount. Currently set to 5 */
        require (msg.value == amount);
        _;
    }

    constructor() public payable
    {
        /** constructor holding results matrix for determining winners */
        results["paper"]["rock"] = 1;
        results["paper"]["paper"] = 0;
        results["paper"]["scissors"] = 2;
        results["rock"]["rock"] = 0;
        results["rock"]["paper"] = 2;
        results["rock"]["scissors"] = 1;
        results["scissors"]["rock"] = 2;
        results["scissors"]["paper"] = 1;
        results["scissors"]["scissors"] = 0;
    }

    function clear_game() public
    {
        player1Hand = "";
        player2Hand = "";
        player1 = 0;
        player2 = 0;
    }

    function totalGameMoney () external view returns (uint Current_Game_Money_On_Contract, uint P)
    {
    }
}
```

[2] only remix transactions, script ▾ Search transactions

The screenshot shows the Remix Solidity IDE interface. The main area displays a Solidity smart contract named 'RockPaperScissors.sol'. The code implements a game where two players can register, show their hand (Rock, Paper, or Scissors), and calculate the winner based on the rules of Rock-Paper-Scissors. The interface includes a code editor with syntax highlighting, a sidebar for contract settings like 'Envir', 'Accor', 'Gas l', and 'Value', and a transaction history section at the bottom.

```
function totalGameMoney () external view returns (uint Current_Game_Money_On_Contract, uint player1Balance, uint player2Balance) {
    return (address(this).balance, address(player1).balance, address(player2).balance);
}

function register() public payable registration_Doesnt_Exist notPoor(5 ether) returns (bool success) {
    /* initial registration. first player to register is player1, second player to register is player2 */
    if (player1 == 0) {
        player1 = msg.sender;
        // emit PlayerPhase(msg.sender,1);
        return true;
    } else if (player2 == 0) {
        player2 = msg.sender;
        // emit PlayerPhase(msg.sender,1);
        return true;
    }
    return false;
}

function show_hand(string hand) public registered returns (bool x)
{
    /* stores the players hand in easy readable format if encryption and hand match */
    if (msg.sender == player1)
    {
        player1Hand = hand;
        return true;
    }
    if (msg.sender == player2)
    {
        player2Hand = hand;
        return true;
    }
    return false;
}

function calculateScore() public registered returns (string winner_Is)
{
    if (bytes(player1Hand).length != 0 && bytes(player2Hand).length != 0) // This will trigger
    {
        int winner=0;
        winner = results[player1Hand][player2Hand];
        if (winner == 1){
            /* player 1 rocks and gets the winnings */
            player1.transfer(address(this).balance);
            the_Winner_Is="player 1";
        }
        else if (winner == 2){
            /* player 2 rocks and gets the winnings */
            player2.transfer(address(this).balance);
            the_Winner_Is="player 2";
        }
        else if (winner == 3){
            /* draw */
            the_Winner_Is="draw";
        }
    }
}
```

The screenshot shows the Remix Solidity IDE interface. The main area displays a Solidity contract named `RockPaperScissors.sol`. The code implements a game where two players (player1 and player2) can show their hand (Rock, Paper, or Scissors). The contract then calculates the winner based on the rules of the game. If both players show the same hand, it's a draw. The winning player receives the other player's balance. The contract also emits an event `PlayerPhase` when a player shows their hand.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract RockPaperScissors {
    event PlayerPhase(address indexed sender, string hand);

    string public player1Hand;
    string public player2Hand;
    string public the_Winner_Is;

    function show_hand(string memory hand) public registered returns (bool x) {
        if (msg.sender == player1)
        {
            player1Hand = hand;
            return true;
        }
        if (msg.sender == player2)
        {
            player2Hand = hand;
            return true;
        }
        return false;
    }

    function calculateScore() public registered returns (string winner_Is)
    {
        if (bytes(player1Hand).length != 0 && bytes(player2Hand).length != 0) // This will trigger the event
        {
            int winner=0;
            winner = results[player1Hand][player2Hand];
            if (winner == 1){
                /* player 1 rocks and gets the winnings */
                player1.transfer(address(this).balance);
                the_Winner_Is="player 1";
            }
            else if (winner == 2){
                /* player 2 probably didn't pick scissors */
                player2.transfer(address(this).balance);
                the_Winner_Is="player 2";
            }
            else{
                /* wait, nobody won? this game is shit */
                player1.transfer(address(this).balance/2);
                player2.transfer(address(this).balance);
                the_Winner_Is="game draw!";
            }
            return (the_Winner_Is);
        }
        else
            return (the_Winner_Is);
    }
}
```

The screenshot shows the Remix Solidity IDE interface. The main window displays the Solidity code for a Rock Paper Scissors game. The code includes functions for initializing player phases, showing hands, and calculating scores. A static analysis warning is visible, stating that 13 warnings were raised. To the right, a vertical banner with the name "MRUDHULASHENAVA" is displayed.

```
ContractDefinition RockPaperScissors 0 reference(s) ▾
brows...
RockP... 72 // emit PlayerPhase(msg.sender,1);
ballot... 73 return true;
ballot... 74 }
ballot... 75 }
ballot... 76 }
hello.. 77 }
max.s... 78 function show_hand(string hand) public registered re...
maxin... 79 {
maxin... 80     /* stores the players hand in easy readable for...
rockP... 81 if (msg.sender == player1)
rockP... 82 {
rockP... 83     player1Hand = hand;
rockP... 84     return true;
rockP... 85 }
rockP... 86 }
rockP... 87 if (msg.sender == player2)
rockP... 88 {
rockP... 89     player2Hand = hand;
rockP... 90     return true;
rockP... 91 }
rockP... 92 }
rockP... 93 }
rockP... 94 }
rockP... 95 }
rockP... 96 }
rockP... 97 }
rockP... 98 }
rockP... 99 }
rockP... 100 }
rockP... 101 }
rockP... 102 }
rockP... 103 }
rockP... 104 }
rockP... 105 }

[2] only remix transactions, script
```

We write the code for rock paper scissors game where two players will play the game and initially we set a limit for ethers to be bid by each player which they register with and then show their hands and then we calculate the winner and who the ether goes to. We compile the program.

The screenshot shows the Remix - Solidity IDE interface. On the left, the code editor displays the `RockPaperScissors.sol` file. The code implements a game where two players can register and show their hand (Rock, Paper, Scissors). The deployed contract at address `0xaca3...a733c` has the following details:

ash	55c607d30a097119f99387529fa060d
contract address	0xdcb4977a2078c8ffdf086d18d1f061b6c546222
from	0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to	RockpaperScissors.(constructor)
gas	3000000 gas
transaction cost	1306869 gas
execution cost	939285 gas
hash	0x3bd0b357a2d045d2fd43df7d5b0fe20ee55c607d30a097119f99387529fa060d
input	0x608...d0029
decoded input	{}
decoded output	-

The right panel shows the environment settings (JavaScript VM), account (0xca3...a733c), gas limit (300000), and value (0). It also lists the deployed contract's functions: calculateScore, clear\_game, register, show\_hand, the\_Winner\_Is, and totalGameMoney. A vertical watermark "MRUDHULASHENAVA" is visible on the right side.

We run the program and click on deploy and see in console log that currently no decoded input or output is assigned.

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract RockPaperScissors {
    string player1Hand;
    string player2Hand;
    uint256 Current_Game_Money_On_Contract;
    uint256 Player_1_Account_Balance;
    uint256 Player_2_Account_Balance;

    event PlayerPhase(string hand, bool result);

    function register() public {
        if (msg.sender == player1)
            player1Hand = msg.data;
        else if (msg.sender == player2)
            player2Hand = msg.data;
        else
            emit PlayerPhase(msg.data, false);
    }

    function calculateScore(string memory hand) public {
        if (hand == "Rock") {
            if (player1Hand == "Rock")
                Current_Game_Money_On_Contract += 1000000000000000000;
            else if (player1Hand == "Paper")
                Current_Game_Money_On_Contract -= 1000000000000000000;
            else if (player1Hand == "Scissors")
                Current_Game_Money_On_Contract += 500000000000000000;
        } else if (hand == "Paper") {
            if (player1Hand == "Rock")
                Current_Game_Money_On_Contract -= 1000000000000000000;
            else if (player1Hand == "Paper")
                Current_Game_Money_On_Contract += 1000000000000000000;
            else if (player1Hand == "Scissors")
                Current_Game_Money_On_Contract -= 500000000000000000;
        } else if (hand == "Scissors") {
            if (player1Hand == "Rock")
                Current_Game_Money_On_Contract += 500000000000000000;
            else if (player1Hand == "Paper")
                Current_Game_Money_On_Contract -= 500000000000000000;
            else if (player1Hand == "Scissors")
                Current_Game_Money_On_Contract += 1000000000000000000;
        }
    }

    function clear_game() public {
        Current_Game_Money_On_Contract = 0;
        Player_1_Account_Balance = 0;
        Player_2_Account_Balance = 0;
    }

    function show_hand(string memory hand) public registered returns (bool) {
        if (msg.sender == player1)
            player1Hand = hand;
        else if (msg.sender == player2)
            player2Hand = hand;
        else
            return false;
    }

    function the_Winner_Is(string memory hand) public {
        if (hand == "Rock") {
            if (player1Hand == "Rock")
                Current_Game_Money_On_Contract += 1000000000000000000;
            else if (player1Hand == "Paper")
                Current_Game_Money_On_Contract -= 1000000000000000000;
            else if (player1Hand == "Scissors")
                Current_Game_Money_On_Contract += 500000000000000000;
        } else if (hand == "Paper") {
            if (player1Hand == "Rock")
                Current_Game_Money_On_Contract -= 1000000000000000000;
            else if (player1Hand == "Paper")
                Current_Game_Money_On_Contract += 1000000000000000000;
            else if (player1Hand == "Scissors")
                Current_Game_Money_On_Contract -= 500000000000000000;
        } else if (hand == "Scissors") {
            if (player1Hand == "Rock")
                Current_Game_Money_On_Contract += 500000000000000000;
            else if (player1Hand == "Paper")
                Current_Game_Money_On_Contract -= 500000000000000000;
            else if (player1Hand == "Scissors")
                Current_Game_Money_On_Contract += 1000000000000000000;
        }
    }

    function totalGameMoney() public view returns (uint256) {
        return Current_Game_Money_On_Contract;
    }

    function Player_1_Account_Balance() public view returns (uint256) {
        return Player_1_Account_Balance;
    }

    function Player_2_Account_Balance() public view returns (uint256) {
        return Player_2_Account_Balance;
    }
}

```

**Transactions recorded:**

h	7d30a097119f99387529fa0600
contract address	0xdc04977a2078c8ffdf086d618d1f961b6c546222
from	0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to	RockPaperScissors.(constructor)
gas	3000000 gas
transaction cost	1306869 gas
execution cost	939285 gas
hash	0x3bd0b357a2d045d2fd43df7d5b0fe20ee55c607d30a097119f99387529fa0600
input	0x608...d0029
decoded input	{}
decoded output	-
logs	[]
value	0 wei

**Deployed Contracts**

- RockPaperScissors at 0xdc0...46222 (memory)
  - calculateScore
  - clear\_game
  - register
  - show\_hand string hand
  - the\_Winner\_Is
    - 0: string: nobody wins yet
  - totalGameMoney

0: uint256: Current\_Game\_Money\_On\_Contract 0  
1: uint256: Player\_1\_Account\_Balance 0  
2: uint256: Player\_2\_Account\_Balance 0

We then check the totalGameMoney function, this function will show how many wei is left after each player has bid and how much ether the contract has. Currently we see that there are no transactions. Hence, balance is 0 throughout.

M  
R  
U  
D  
H  
U  
L  
A

### Remix - Solidity IDE

browser/RockPaperScissors.sol

```
// emit PlayerPhase(msg.sender,1);
    return true;
}
return false;
}

function show_hand(string hand) public registered returns (bool)
{
    /* stores the players hand in easy readable format if entered */
    if (msg.sender == player1)
    {
        player1Hand = hand;
        return true;
    }
    if (msg.sender == player2)
    {
        player2Hand = hand;
        return true;
    }
    return false;
}
```

[2] only remix transactions, script

from	0x14723a09acff6d2a60dcdf7aa4aff308fddc16 0c
to	RockPaperScissors.totalGameMoney() 0xdc0 4977a2078c8ffd086d61d1f9e1b6c546222
transaction cost	23275 gas (Cost only applies when called by a contract)
execution cost	2003 gas (Cost only applies when called by a contract)
hash	0xd9c4d2dc99676b6b8b145bf66b0f0ad196a80 7ed21b5e3e5c4acdcbd0c64710
input	0x387...6a8c5
decoded input	{}
decoded output	{ "0": "uint256: Current_Game_Money_On_Contract 50000000000000000000", "1": "uint256: Player_1_Account_Balance 94999999999996957400", "2": "uint256: Player_2_Account_Balance 0" }

Account 0x4b0...4d2db (94.9999999999999571)

Gas limit 3000000

Value 0 ether

RockPaperScissors

Deploy

Load contract from Address At Address

Transactions recorded: (3)

Deployed Contracts

RockPaperScissors at 0xdc0...46222 (memory)

calculateScore

clear\_game

register

show\_hand string hand

the\_Winner\_Is

0: string: nobody wins yet

totalGameMoney

0: uint256: Current\_Game\_Money\_On\_Contract 10000000000000000000

1: uint256: Player\_1\_Account\_Balance 94999999999999957400

2: uint256: Player\_2\_Account\_Balance 94999999999996957132

Then we register each player with 5 ether each. And then we check the balance we see that 5 ethers have been reduced from 100 ethers and the balance is shown. Currently contract will have 10 ethers, 5 each from the two players playing.

The screenshot shows the Remix - Solidity IDE interface. On the left, there is a vertical watermark reading "MRUDHULASHENAVA". The main window displays the Solidity code for the "RockPaperScissors" contract. The code includes functions for emitting player phases, showing hands, and calculating scores. In the environment settings, the account is set to 0x147...c160c with a gas limit of 3000000 and a value of 0 ether. A deployed contract "RockPaperScissors at 0xbde...4ddc9 (memory)" is listed with methods: calculateScore, clear\_game, register, and show\_hand. The show\_hand method has a parameter "hand" of type string and a "transact" button. Below the transaction details, it shows the output: "the\_Winner\_Is 0: string: nobody wins yet" and "totalGameMoney".

```

// emit PlayerPhase(msg.sender,1);
return true;
}
return false;
}

function show_hand(string hand) public registered returns (bool)
{
    /* stores the players hand in easy readable format if enc
    if (msg.sender == player1)
    {
        player1Hand = hand;
        return true;
    }
    if (msg.sender == player2)
    {
        player2Hand = hand;
        return true;
    }
    return false;
}

```

	[2] only remix transactions, script
transaction hash	0x92abd567bdc274101b8fe53dcec6b1a82b113ee a5ad8604bb4b40b1272affa81
from	0x14723a09acff6d2a60dcdf7aa4aff308fddc160 c
to	RockPaperScissors.show_hand(string) 0xbde 95422681e4c3984635af2f2f35f8c44a4ddc9
gas	3000000 gas
transaction cost	44202 gas
execution cost	22162 gas
hash	0x92abd567bdc274101b8fe53dcec6b1a82b113ee a5ad8604bb4b40b1272affa81
input	0x610...00000
decoded input	{         "string hand": "rock"     }
decoded output	{         "0": "bool: x true"     }

We initiate the hand for player 1 as shown in the above screenshot player 1 plays rock as shown in console log above and we click transact for the hand to get stored. When we check for the winner it says there are no winners yet.

The screenshot shows the Remix Solidity IDE interface. On the left, there is a vertical watermark reading "MRUDHULASHENAVA". The main window has a title bar "Remix - Solidity IDE" and a tab "browser/RockPaperScissors.sol". The code editor contains the following Solidity code:

```

72     // emit PlayerPhase(msg.sender,1);
73     return true;
74   }
75   return false;
76 }
77
78 function show_hand(string hand) public registered returns (bool)
79 {
80   /** stores the players hand in easy readable format if enc
81   if (msg.sender == player1)
82   {
83     player1Hand = hand;
84     return true;
85   }
86   if (msg.sender == player2)
87   {
88     player2Hand = hand;
89     return true;
90   }
91   return false;
92 }
93
94 }
```

The right side of the interface includes settings for "Environment" (JavaScript VM), "Account" (0x4b0...4d2db), "Gas limit" (3000000), and "Value" (0 ether). Below these are buttons for "Deploy" and "Load contract from Address At Address". A section titled "Transactions recorded: 5" shows a table of transaction details:

from	0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2d b
to	RockPaperScissors.show_hand(string) 0x54aab84579861088ee6ef8219d05116328a15eb
gas	3000000 gas
transaction cost	44266 gas
execution cost	22162 gas
hash	0x5a0632eb8da2ab39e1b5e49fd76751737701789 d4adb5fa29af5088fa2f678a9
input	0x610...00000
decoded input	{ "string hand": "paper" }
decoded output	{ "0": "bool: x true" }
logs	[]

The "Deployed Contracts" section shows the "RockPaperScissors" contract at address 0x54aab84579861088ee6ef8219d05116328a15eb, with methods: calculateScore, clear\_game, register, and show\_hand. The "show\_hand" method has a parameter "hand: string" and a "transact" button. The "Logs" section shows an empty array. The total game money is listed as 0: uint256: Current\_Game\_Money\_On\_Contract 5000000000000000000.

Then we initiate the hand for player 2. Player 2 plays paper as shown in console log. Then we click on transact to store player 2's hand.

The screenshot shows the Remix - Solidity IDE interface. On the left, there is a vertical watermark with the letters M, R, U, D, H, U, L, A. The main window displays the Solidity code for the `RockPaperScissors.sol` contract. The code includes functions for calculating scores based on player hands and emitting events. The right side of the interface shows deployment settings (Environment: JavaScript VM, Account: 0x4b0...4d2db, Gas limit: 3000000, Value: 0 ether), a dropdown for the deployed contract (`RockPaperScissors`), and a section for recorded transactions. Below these are sections for deployed contracts and transaction details.

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract RockPaperScissors {
    event PlayerPhase(address indexed sender, uint8 hand);
    event GameScore(uint8 player1Score, uint8 player2Score);

    uint8 public player1Hand;
    uint8 public player2Hand;
    uint8 public currentGameScore;
    uint8 public totalGameMoney;

    function calculateScore(uint8 player1Hand, uint8 player2Hand) private pure returns (uint8, uint8) {
        if (player1Hand == player2Hand) {
            return (0, 0);
        } else if ((player1Hand == 0 & player2Hand == 1) || (player1Hand == 1 & player2Hand == 2)) {
            return (1, 0);
        } else {
            return (0, 1);
        }
    }

    function register() public {
        require(currentGameScore == 0, "Game is already in progress");
        currentGameScore = 0;
        totalGameMoney = 0;
    }

    function clear_game() public {
        currentGameScore = 0;
        totalGameMoney = 0;
    }

    function show_hand(string memory hand) public registered returns (bool) {
        if (msg.sender == player1) {
            player1Hand = uint8(keccak256(hand).mod(3));
            emit PlayerPhase(msg.sender, 1);
            return true;
        }
        return false;
    }

    function calculateScore() public view returns (uint8, uint8) {
        return calculateScore(player1Hand, player2Hand);
    }

    function the_Winner_Is() public view returns (string memory) {
        if (currentGameScore == 1) {
            return "Player 1 Wins";
        } else if (currentGameScore == 2) {
            return "Player 2 Wins";
        } else {
            return "It's a Tie";
        }
    }

    function totalGameMoney() public view returns (uint256) {
        return totalGameMoney;
    }
}

```

**Deployed Contracts**

- RockPaperScissors at 0x54aa...a15eb (memory)**
  - calculateScore**
  - clear\_game**
  - register**
  - show\_hand**
    - hand: string
    - transact
  - the\_Winner\_Is**
  - totalGameMoney**

**Transactions recorded:** 6

from	0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2d
to	RockPaperScissors.calculateScore() 0x54aaab84579861088ee6ef8219d05116328a15eb
gas	3000000 gas
transaction cost	44199 gas
execution cost	22927 gas
hash	0x2c90e5d257e36295a130c22e7c65decacf433584adfbac2c8246dbae9ec5e9cff
input	0x304...e05bb
decoded input	{}
decoded output	{             "0": "string: winner_Is player 2"         }
logs	[]
value	0 wei

Then we click on calculate score this function calculates from the matrix and checks which condition either players are in. when we check the console log for the calculateScore we can see the winner is specified in decoded output.

The screenshot shows the Remix - Solidity IDE interface. On the left, there is a vertical watermark reading "MRUDHULASHENAVA". The main window displays the Solidity code for the "RockPaperScissors" contract. The code includes functions for player registration, hand selection, and score calculation. A transaction history table is shown below the code editor, detailing a recent call to the "the\_Winner\_Is()" function. The right side of the interface shows the deployed contract details, including its address (0x54a...a15eb), methods like calculateScore, clear\_game, register, and show\_hand, and the totalGameMoney function which returns the current game money distribution.

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract RockPaperScissors {
    mapping(address => string) public registered;
    string[] public hands = ["Rock", "Paper", "Scissors"];
    uint256 public totalGameMoney = 10 ether;
    uint256 public Current_Game_Money_On_Contract = 0;
    uint256 public Player_1_Account_Balance = 0;
    uint256 public Player_2_Account_Balance = 0;

    event PlayerPhase(string hand, uint256 value);
    event GameReset();

    constructor() {
        Current_Game_Money_On_Contract = totalGameMoney;
        Player_1_Account_Balance = totalGameMoney / 2;
        Player_2_Account_Balance = totalGameMoney / 2;
    }

    modifier registered() {
        require(registered[msg.sender] != "", "Player not registered");
        _;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner can call this function");
        _;
    }

    function register(string memory hand) public registered returns (bool) {
        registered[msg.sender] = hand;
        emit PlayerPhase(hand, 1);
        return true;
    }

    function show_hand(string memory hand) public registered returns (bool) {
        /* stores the players hand in easy readable format if enc */
        if (msg.sender == player1) {
            player1Hand = hand;
            return true;
        }
        if (msg.sender == player2) {
            player2Hand = hand;
            return true;
        }
        return false;
    }

    function calculateScore(string memory hand1, string memory hand2) private pure returns (uint256) {
        if (hand1 == hand2) {
            return 0;
        }
        if (hand1 == "Rock" & hand2 == "Scissors") {
            return 1;
        }
        if (hand1 == "Scissors" & hand2 == "Paper") {
            return 1;
        }
        if (hand1 == "Paper" & hand2 == "Rock") {
            return 1;
        }
        if (hand1 == "Rock" & hand2 == "Paper") {
            return -1;
        }
        if (hand1 == "Scissors" & hand2 == "Rock") {
            return -1;
        }
        if (hand1 == "Paper" & hand2 == "Scissors") {
            return -1;
        }
    }

    function clear_game() public onlyOwner {
        Current_Game_Money_On_Contract = 0;
        Player_1_Account_Balance = 0;
        Player_2_Account_Balance = 0;
        emit GameReset();
    }

    function register(string memory hand) public onlyOwner {
        registered[msg.sender] = hand;
    }

    function show_hand(string memory hand) public onlyOwner {
        player1Hand = hand;
    }

    function the_Winner_Is() public view returns (string memory) {
        if (player1Hand == "Rock" & player2Hand == "Scissors") {
            return "player 1";
        }
        if (player1Hand == "Scissors" & player2Hand == "Paper") {
            return "player 1";
        }
        if (player1Hand == "Paper" & player2Hand == "Rock") {
            return "player 1";
        }
        if (player1Hand == "Rock" & player2Hand == "Paper") {
            return "player 2";
        }
        if (player1Hand == "Scissors" & player2Hand == "Rock") {
            return "player 2";
        }
        if (player1Hand == "Paper" & player2Hand == "Scissors") {
            return "player 2";
        }
        return "tie";
    }

    function totalGameMoney() public view returns (uint256) {
        return Current_Game_Money_On_Contract;
    }

    function withdraw() public {
        if (msg.sender == player1) {
            msg.sender.transfer(Player_1_Account_Balance);
            Player_1_Account_Balance = 0;
        }
        if (msg.sender == player2) {
            msg.sender.transfer(Player_2_Account_Balance);
            Player_2_Account_Balance = 0;
        }
    }
}

```

transaction hash	0xfe0330a852835e5c1e26db6c4fbff19566c7e8a06ddf69289d8f554c90c27e5
from	0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db
to	RockPaperScissors.the_Winner_Is() 0x54aaaabb84579861088ee6ef8219d05116328a15eb
transaction cost	22800 gas (Cost only applies when called by a contract)
execution cost	1528 gas (Cost only applies when called by a contract)
hash	0xfe0330a852835e5c1e26db6c4fbff19566c7e8a06ddf69289d8f554c90c27e5
input	0x8ae...65367
decoded input	{}
decoded output	{"0": "string: player 2"}

Now when we click on the totalGameMoney function we see that the winner gets all the ether that contract had. The balance against the contract is 0 ethers, against player 1 is still the total – 5 ethers and players 2 gets 10 complete ethers.

The screenshot shows the Remix Solidity IDE interface. On the left, there is a vertical watermark with the letters "M", "R", "U", "D", "H", "U", "L", "A". The main window title is "Remix - Solidity IDE" and the file name is "browser/RockPaperScissors.sol". The code editor displays the following Solidity code:

```

72     // emit PlayerPhase(msg.sender,1);
73     return true;
74   }
75   return false;
76 }
77
78 function show_hand(string hand) public registered returns (boo
79 {
80   /* stores the players hand in easy readable format if enc
81   if (msg.sender == player1)
82   {
83     player1Hand = hand;
84     return true;
85   }
86   if (msg.sender == player2)
87   {
88     player2Hand = hand;
89     return true;
90   }
91   return false;
92 }
93
94
95

```

The right side of the interface includes tabs for "Compile", "Run", "Settings", "Analysis", "Debugger", and "Support". The "Run" tab is active, showing a gas limit of 3000000, a value of 0, and an ether input field. A dropdown menu shows "RockPaperScissors" and a "Deploy" button. Below that is a "Transactions recorded: 7" section and a "Deployed Contracts" section listing "RockPaperScissors at 0x54a...a15eb (memory)". Under the deployed contract, there are buttons for "calculateScore", "clear\_game", "register", and "show\_hand". The "show\_hand" section has a parameter "hand: string" and a "transact" button. Transaction history shows several calls to "the\_Winner\_Is" from the same address, with the most recent one being a call to "RockPaperScissors.the\_Winner\_Is()". The "totalGameMoney" section shows three uint256 values: 0: Current\_Game\_Money\_On\_Contract 0, 1: Player\_1\_Account\_Balance 0, and 2: Player\_2\_Account\_Balance 0.

We have another option that is to clear game when we click on it the balance gets cleared out and zero's out as shown above.