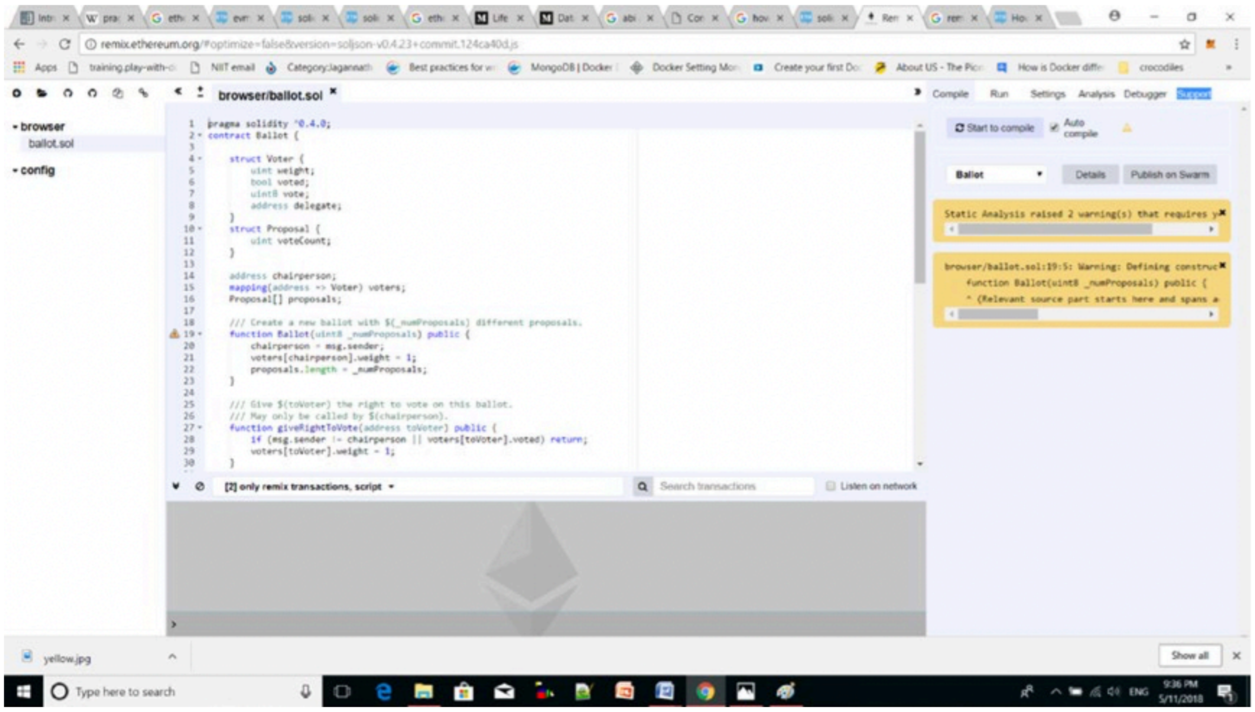
Using remix:

Good tutorial:

https://www.sitepoint.com/remix-smart-contracts-ethereum-blockchain/

Remix Browser

The best IDE for learning Solidity is Remix. It is an online browser-based IDE available at http://remix.ethereum.org that comes with a source editor and a file manager as well as with compiling, deploying, and debugging options. While developing, you have to make sure you are connected to the Internet.



```
pragma solidity ^0.4.0;
contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        uint8 vote;
        address delegate;
    }
    struct Proposal {
        uint voteCount;
    }

    address chairperson;
    mapping(address => Voter) voters;
    Proposal[] proposals;

    /// Create a new ballot with $(_numProposals) different proposals.
    function Ballot(uint8 _numProposals) public {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;
        proposals.length = _numProposals;
    }

    /// Give $(toVoter) the right to vote on this ballot.
    /// May only be called by $(chairperson).
    function giveRightToVote(address toVoter) public {
        if (msg.sender != chairperson || voters[toVoter].voted) return;
        voters[toVoter].weight = 1;
    }
```

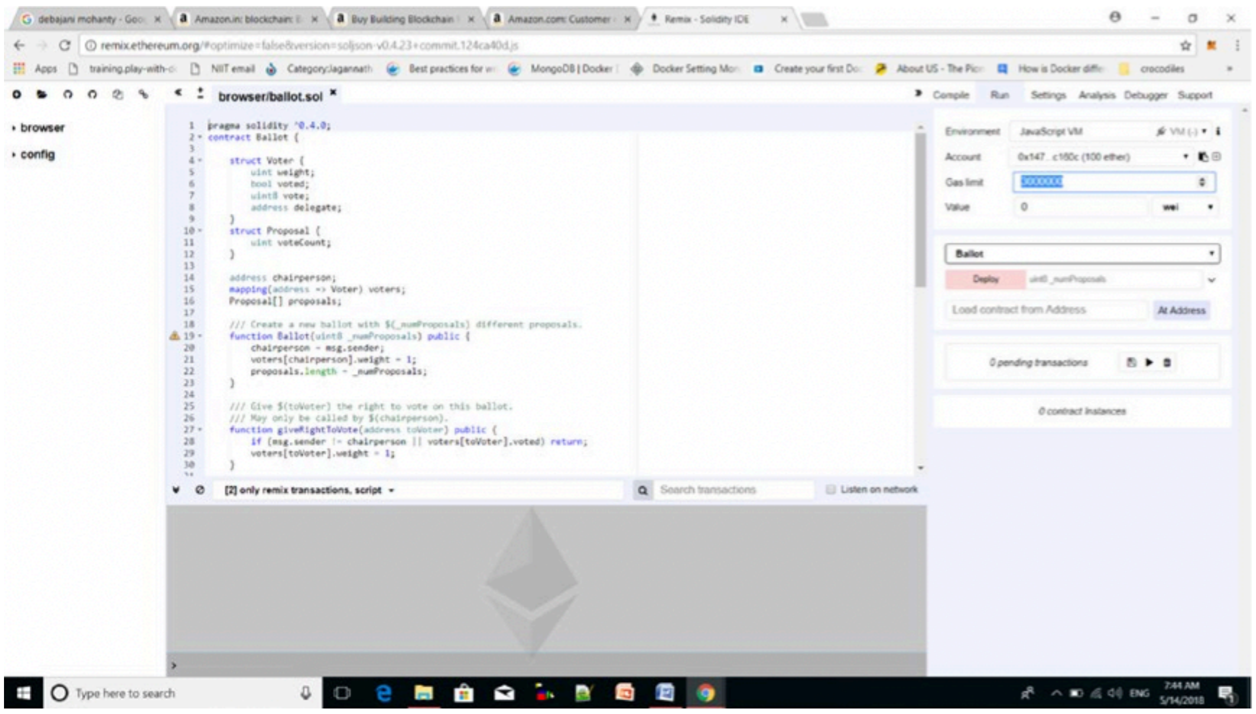
```
Static Analysis raised 2 warning(s) that requires y
```

```
browser/ballot.sol:19:5: Warning: Defining construc
    function Ballot(uint8 _numProposals) public {
    ^ (Relevant source part starts here and spans a
```

Using Remix:

This IDE comes with a default contract named **ballet.sol**, which I will discuss later in the chapter in detail. For the time being, you can click this file on the left side of the screen if it's not open already and then expand it to browse through it. You can also click the + link on the menu to create a new contract. When you write your contract, you can either compile your contract manually by clicking the **"Start to compile" button** on right side of the Compile tab or click the **"Auto compile"** check box to get it done on the fly.

Remix will hold all of your smart contracts in browser unless you delete them



```
pragma solidity ^0.4.0;
contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        uint8 vote;
        address delegate;
    }
    struct Proposal {
        uint voteCount;
    }

    address chairperson;
    mapping(address => Voter) voters;
    Proposal[] proposals;

    /// Create a new ballot with $(_numProposals) different proposals.
    function Ballot(uint8 _numProposals) public {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;
        proposals.length = _numProposals;
    }

    /// Give $(toVoter) the right to vote on this ballot.
    /// May only be called by $(chairperson).
    function giveRightToVote(address toVoter) public {
        if (msg.sender != chairperson || voters[toVoter].voted) return;
        voters[toVoter].weight = 1;
    }
```

Environment: JavaScript VM
Account: 0x147...c160c (100 ether)
Gas limit: 3000000
Value: 0 wei
Ballot

Registering a contract on the blockchain involves creating a special transaction whose destination is the address 0x0000000000000000000000000000000000000000, also known as the zero address. The zero address is a special address that tells the Ethereum blockchain that you want to register a contract. Fortunately, the Remix IDE will handle all of that for you and send the transaction to MetaMask.

First, switch to the Run tab and select Injected Web3 in the Environment drop-down selection box. This connects the Remix IDE to the MetaMask wallet, and through MetaMask to the Ropsten test network.

Note: You may simulation the environment for quick testing using Javascript VM or connect to injected web3 or setup a web3 Provider

There are accounts for testing in the dropdown if you leave it to Javascript VM

If you connect to a network, you will need to have Truffle running in the background

Or ganache-cli from command line.

Truffle and ganache give you a sample sandbox blockchain for testing.

When you connect to that network, your accounts will be shown in the dropdown

Address

An address is a type that saves an account address on an Ethereum node. Addresses can be useful, as you will learn in case studies and examples. Using an address's balance and transfer functions, you can transfer an amount from one to another address on Ethereum nodes.

Setting up for Deployment:

Deploying Smart Contracts

Now that you are proficient in smart contract development using Solidity, let's deploy a contract on an Ethereum network. Deployment can be done in many ways.

- *Deploy in Remix with the JavaScript VM*: This is something you have already done.

- *Ganache*: Formerly known as TestRPC, Ganache is a local private setup running on your machine for development and unit testing.

- *Ropsten/Kovan/Rinkeby test network and MetaMask*: These are Ethereum clients used for functional testing.

- *Truffle Suite*: Truffle Suite is a development and testing framework for smart contracts on Ethereum. This will be discussed later in the book in detail.

Local Ethereum Testing with Ganache

TestRPC is a Node.js-based Ethereum client for client testing and development. It offers a private blockchain network that runs only on your own machine without connecting to any other node in the Ethereum network; however, it imitates the properties of a test or live Ethereum network. Also, it's a fast, flexible, painless way to emulate calls to the blockchain without the overhead of running an actual Ethereum node. It comes with ten accounts with their own private keys that you can use as per your needs.

To run ganache-cli, follow these instructions:

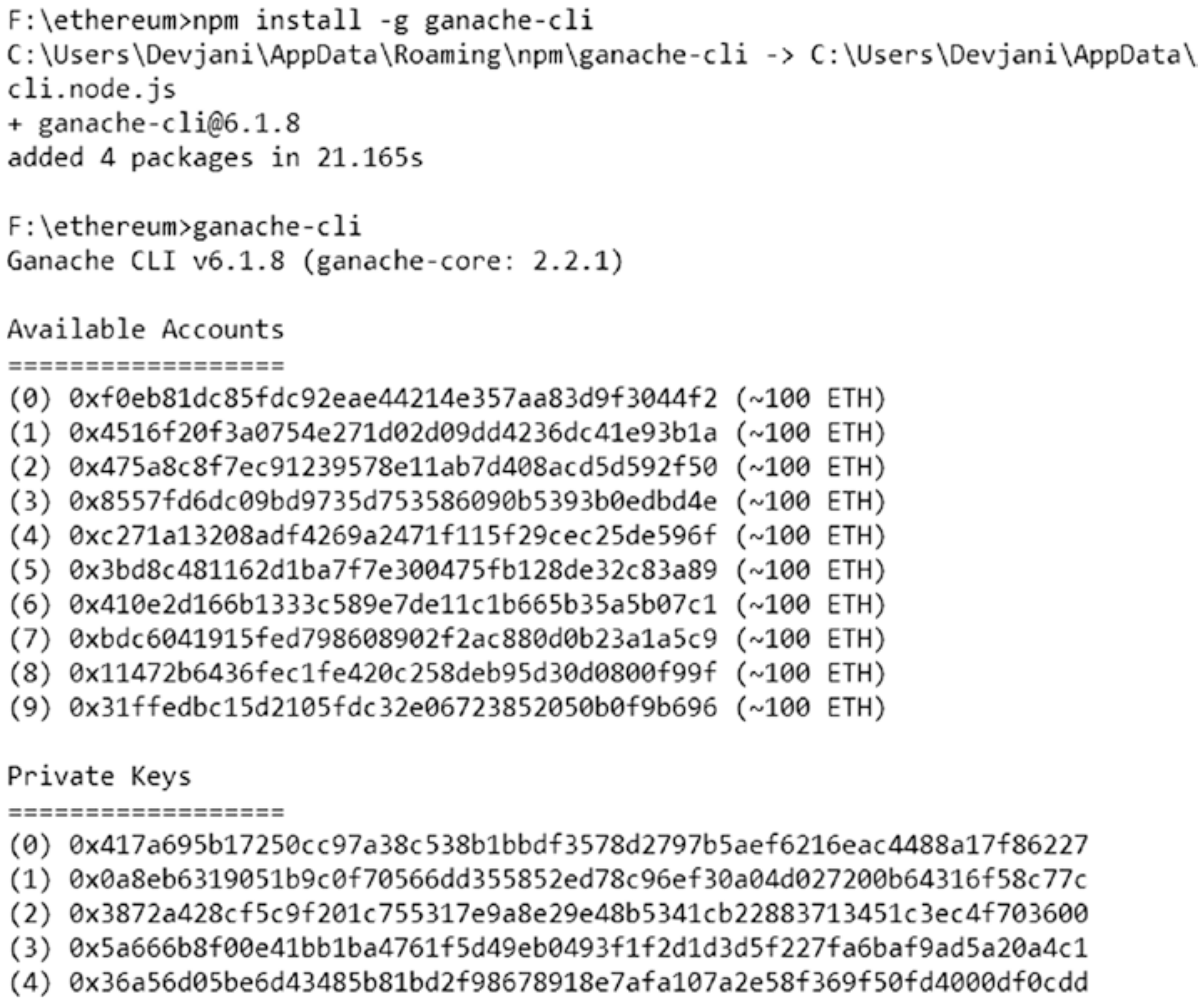
1. Install a node on your machine. Check the version with the following commands:
   node –v npm -v

2. Use NPM to install ganache-cli. npm install -g ganache-cli

3. Once ganache-cli is installed, you can run the following from the command line to run the test environment: ganache-cli

```
F:\ethereum>ganache-cli
Ganache CLI v6.1.8 (ganache-core: 2.2.1)
Available Accounts
==================
(0) 0xc1039b33cf5736bcffab1eee983af7bc1b49c979 (~100 ETH)
(1) 0xa7dd571bcc652d74432d1ea1b5a830aba775286f (~100 ETH)
(2)0x4b02315fe9b74cfdbd7b1a8f6643951bc81f62fc (~100 ETH)
```

Deploying Smart ContraCtS example:



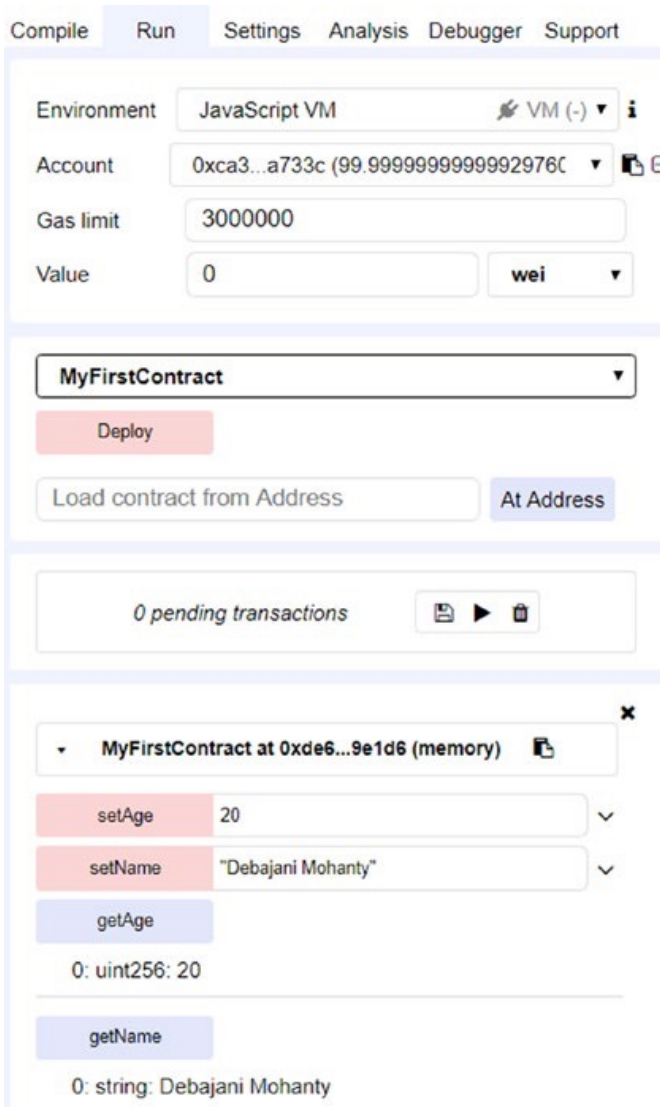
```
F:\ethereum>npm install -g ganache-cli
C:\Users\Devjani\AppData\Roaming\npm\ganache-cli -> C:\Users\Devjani\AppData\
cli.node.js
+ ganache-cli@6.1.8
added 4 packages in 21.165s

F:\ethereum>ganache-cli
Ganache CLI v6.1.8 (ganache-core: 2.2.1)

Available Accounts
==================
(0) 0xf0eb81dc85fdc92eae44214e357aa83d9f3044f2 (~100 ETH)
(1) 0x4516f20f3a0754e271d02d09dd4236dc41e93b1a (~100 ETH)
(2) 0x475a8c8f7ec91239578e11ab7d408acd5d592f50 (~100 ETH)
(3) 0x8557fd6dc09bd9735d753586090b5393b0edbd4e (~100 ETH)
(4) 0xc271a13208adf4269a2471f115f29cec25de596f (~100 ETH)
(5) 0x3bd8c481162d1ba7f7e300475fb128de32c83a89 (~100 ETH)
(6) 0x410e2d166b1333c589e7de11c1b665b35a5b07c1 (~100 ETH)
(7) 0xbdc6041915fed798608902f2ac880d0b23a1a5c9 (~100 ETH)
(8) 0x11472b6436fec1fe420c258deb95d30d0800f99f (~100 ETH)
(9) 0x31ffedbc15d2105fdc32e06723852050b0f9b696 (~100 ETH)

Private Keys
==================
(0) 0x417a695b17250cc97a38c538b1bbdf3578d2797b5aef6216eac4488a17f86227
(1) 0x0a8eb6319051b9c0f70566dd355852ed78c96ef30a04d027200b64316f58c77c
(2) 0x3872a428cf5c9f201c755317e9a8e29e48b5341cb22883713451c3ec4f703600
(3) 0x5a666b8f00e41bb1ba4761f5d49eb0493f1f2d1d3d5f227fa6baf9ad5a20a4c1
(4) 0x36a56d05be6d43485b81bd2f98678918e7afa107a2e58f369f50fd4000df0cdd
```

4. This provides you with 10 different accounts, each with 100 ethers and private keys, along with a local server running on localhost:8545 or 127.0.0.1:8545.

5. Now open your Remix browser and paste the StudentDetails.sol Solidity contract shown in Listing 4-2 into the browser.

Deploying Contracts:

Once your contract is compiled successfully, you can click the Run tab on the right side and choose an option from the drop-down. Choose JavaScript VM, which is the default mode when you don't use Remix with Mist or MetaMask



Environment JavaScript VM VM (-)
Account 0xca3...a733c (99.999999999992976
Gas limit 3000000
Value 0 wei
MyFirstContract
Load contract from Address
0 pending transactions
MyFirstContract at 0xde6...9e1d6 (memory)
setAge 20
setName "Debajani Mohanty"
0: uint256: 20
0: string: Debajani Mohanty

Click the deploy button…

In some cases, there is a constructor which takes data that you will need to fill in

In the bottom pane…you can see the creation contract after deploying

If you connect to a network and use Metamask, it will pop up with a box for you to approve the transaction.

File Extension:

Every file is saved as a .sol

The file will be available on your Chrome browser unless you either delete it or clear the cache. To avoid accidental deletion of the file from browser storage, you can use Remixd, which enables you to store and sync files in the browser with your local computer.

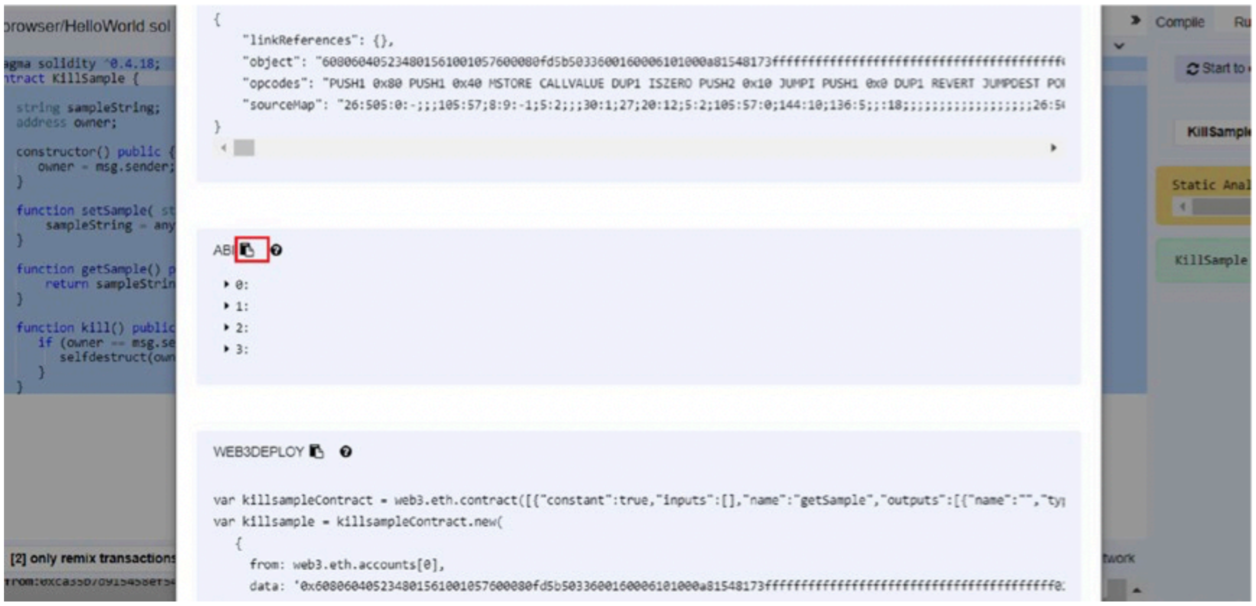
For further details, refer to http:// remix.readthedocs.io/en/latest/ tutorial_remixd_filesystem.html.

Application Binary Interface:

Every contract has an application binary interface (ABI), which is pretty much like an API that works as an interface between the high-level language and the lower-level binary code that gets processed by dumb computers. The ABI consists of the following:

- All function names
- Input and output types of functions
- All event names and their parameters

If you open the Remix browser at http://remix.ethereum.org, open the default ballot.sol, click its Compile tab, and then click the Details button, a pop-up appears. You can find the ABI section under it. Click the rectangular area to copy the ABI.



```
agma solidity ^0.4.18;
ntract KillSample {

  string sampleString;
  address owner;

  constructor() public {
     owner = msg.sender;
  }

  function setSample( st
       sampleString = any
  }

  function getSample() p
       return sampleStrin
  }

  function kill() public
     if (owner == msg.se
        selfdestruct(own
     }
  }
```

```
{
    "linkReferences": {},
    "object": "6080604052348015610010576000080fd5b5033600160006101000a81548173ffffffffffffffffffffffffffffffffffffffff
    "opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST PO
    "sourceMap": "26:505:0:-;;;105:57;8:9:-1;5:2;;;30:1;27;20:12;5:2;105:57:0;144:10;136:5;;:18;;;;;;;;;;;;;;;;;;26:5
}
```

ABI

```
▸ 0:
▸ 1:
▸ 2:
▸ 3:
```

WEB3DEPLOY

```
var killsampleContract = web3.eth.contract([{"constant":true,"inputs":[],"name":"getSample","outputs":[{"name":"","ty
var killsample = killsampleContract.new(
   {
     from: web3.eth.accounts[0],
     data: '0x6080604052348015610010576000080fd5b5033600160006101000a81548173ffffffffffffffffffffffffffffffffffffffff0:
```

Sample ABI file: - looks like below

```
[

   {
       "constant": false,
       "inputs": [],
       "name": "kill",
       "outputs": [],
       "payable": false,
       "stateMutability": "nonpayable",
       "type": "function"
},{

} ],

       "name": "setSample",
       "outputs": [],
       "payable": false,
       "stateMutability": "nonpayable",
```

        "type": "function"
},{

},

"constant": false,
"inputs": [
   {
        "name": "anyString",
        "type": "string"
"inputs": [],
"payable": false,
"stateMutability": "nonpayable",
"type": "constructor"

   {
        "constant": true,
        "inputs": [],
        "name": "getSample",
        "outputs": [
{
"name": "",

            "type": "string"
          }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
}]

Import Statement:

Similar to how we do in Javascript, you can import files to use

Using format

```
import "filename" as symbolName;
```

Version

The first line of code in a Solidity file always starts with a pragma annotation where you fix the compiler to a particular version so that if the version of Solidity gets updated, it will not affect compilation leading to incompatibility issues

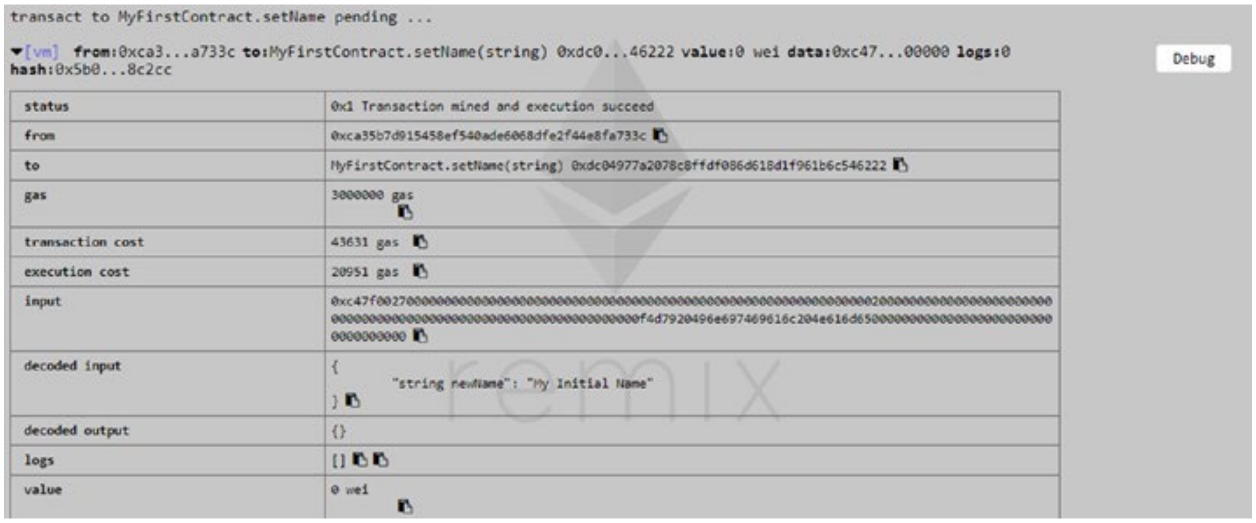
Pragma

```
pragma solidity ^0.4.0;

contract MyContract{
}
```

The Remix Console:

Remix console

Expand each section to check whether there is any issue and also different properties such as status, gas cost, and so on. It also gives you the facility to debug, which I will discuss later in the book.



```
transact to MyFirstContract.setName pending ...

▼[vm] from:0xca3...a733c to:MyFirstContract.setName(string) 0xdc0...46222 value:0 wei data:0xc47...00000 logs:0
hash:0x5b0...8c2cc
```

status	0x1 Transaction mined and execution succeed
from	0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to	MyFirstContract.setName(string) 0xdc04977a2078c8ffdf086d618d1f961b6c546222
gas	3000000 gas
transaction cost	43631 gas
execution cost	20951 gas
input	0xc47f0027000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000000000f4d7920496e697469616c204e616d650000000000000000000000000000000000
decoded input	{ "string newName": "My Initial Name" }
decoded output	{}
logs	[]
value	0 wei

Debugging:

The Transaction section comes with a glide bar that helps you
browse through different parts of the code. You can set breakpoints here by
clicking line numbers on the left and then can step over, forward,
or backward to browse through the code. As per the method you are debugging,
you can see values in the Stack, Memory, and Storage sections