Lab: Browser and Web3.js for Contract Interaction

Prerequisites

1. NodeJS and the Node Package Manager (npm) installed
2. Open Terminal (PowerShell on Windows)
3. Empty Directory
4. Ganache Open

Step by Step Instruction

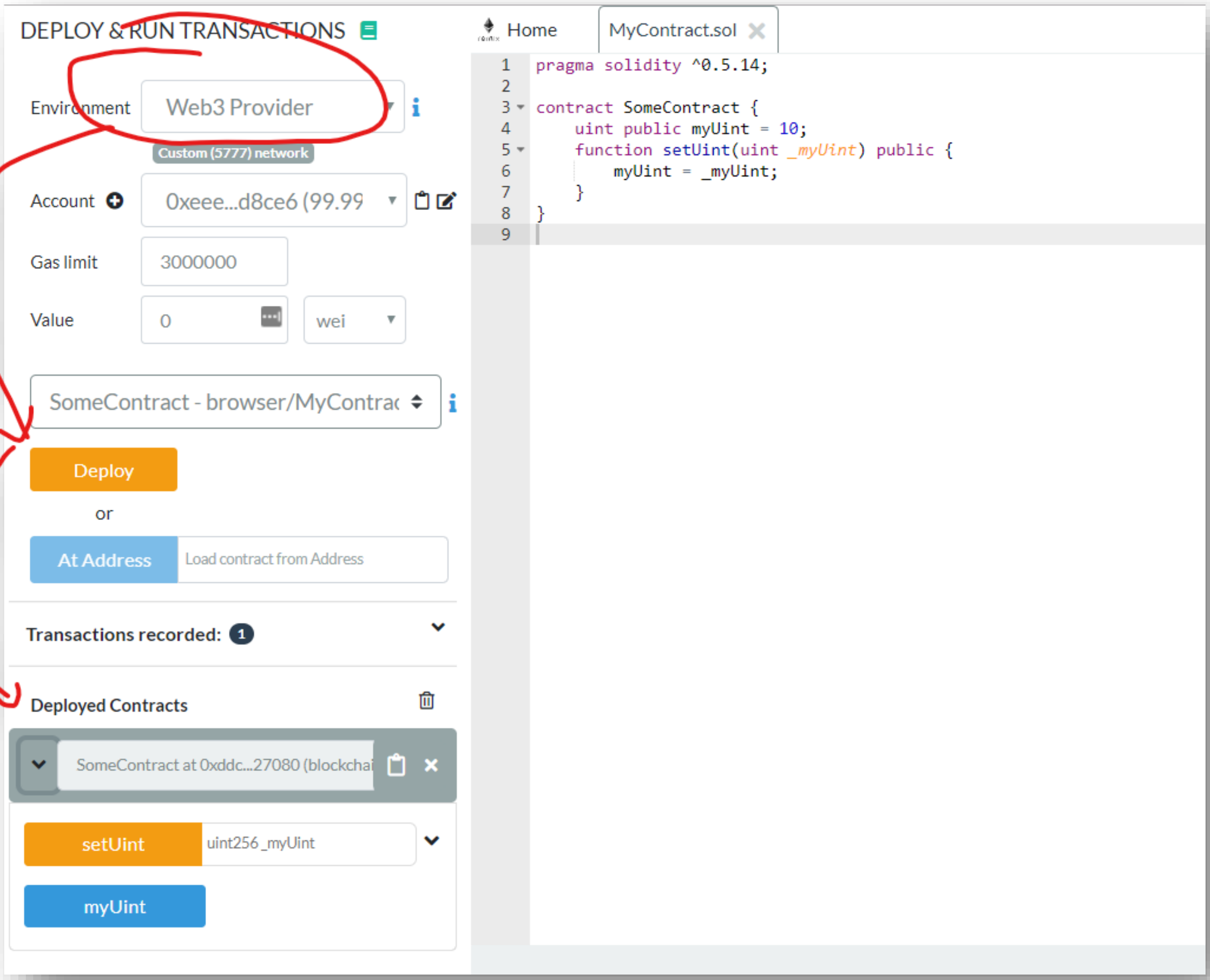
Have this Smart Contract in Remix open

```
pragma solidity ^0.5.14;

contract SomeContract {
    uint public myUint = 10;
    function setUint(uint _myUint) public {
        myUint = _myUint;
    }
}
```

Deploy the Smart Contract with Remix

Choose the Web3-Provider in Remix! Port 7545 if you are using Ganache

Then Deploy the Smart Contract.



Environment: Web3 Provider
Custom (5777) network
Account: 0xeee...d8ce6 (99.99
Gas limit: 3000000
Value: 0 wei
SomeContract - browser/MyContrac
Transactions recorded: 1
Deployed Contracts
SomeContract at 0xddc...27080 (blockchai
uint256 _myUint

```
pragma solidity ^0.5.14;

contract SomeContract {
    uint public myUint = 10;
    function setUint(uint _myUint) public {
        myUint = _myUint;
    }
}
```

Install packaged-version of web3.js

Inside your empty directory install web3js packaged for browser-usage:

“npm install web3.js-browser”

Create a new index.html file

Inside the root folder (where your node_modules folder is) create a new index.html file with the following content:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>My Website</title>
    <meta name='viewport' content='width=device-width, initial-scale=1'>

    <script src='node_modules/web3.js-browser/build/web3.js'></script>
</head>
<body>

</body>
</html>
```

Open the file in your browser (Chrome or Firefox)

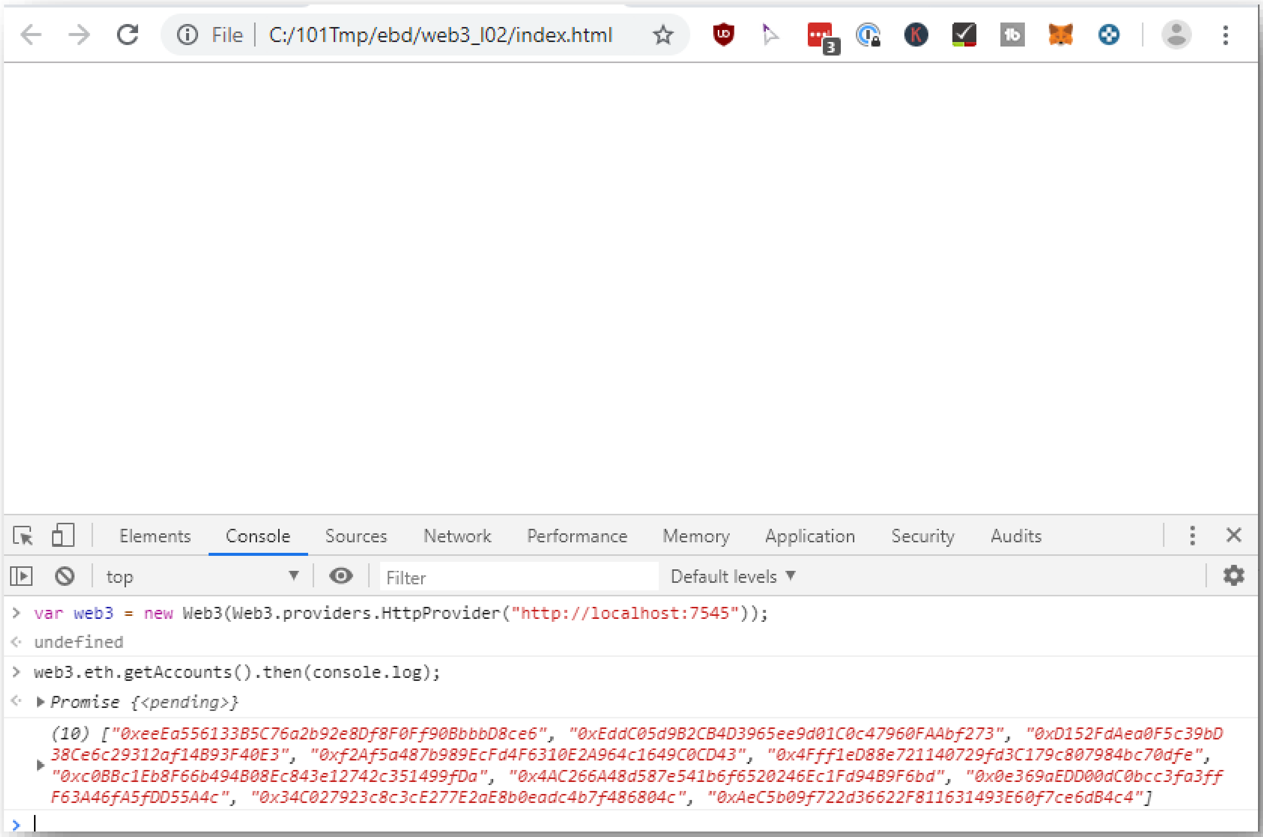
It should be a completely blank page. Then open the developer console in your browser (F12). And enter a few commands to see if you can connect to Ganache:

```
var web3 = new
Web3(Web3.providers.HttpProvider("http://localhost:7545"));
```

```
undefined
```

```
web3.eth.getAccounts().then(console.log);
```

```
Promise {<pending>}

(10) ["0xeeEa556133B5C76a2b92e8Df8F0Ff90BbbbD8ce6",
"0xEddC05d9B2CB4D3965ee9d01C0c47960FAAbf273",
"0xD152FdAea0F5c39bD38Ce6c29312af14B93F40E3",
"0xf2Af5a487b989EcFd4F6310E2A964c1649C0CD43",
"0x4Fff1eD88e721140729fd3C179c807984bc70dfe",
"0xc0BBc1Eb8F66b494B08Ec843e12742c351499fDa",
"0x4AC266A48d587e541b6f6520246Ec1Fd94B9F6bd",
"0x0e369aEDD00dC0bcc3fa3ffF63A46fA5fDD55A4c",
"0x34C027923c8c3cE277E2aE8b0eadc4b7f486804c",
"0xAeC5b09f722d36622F811631493E60f7ce6dB4c4"]
```

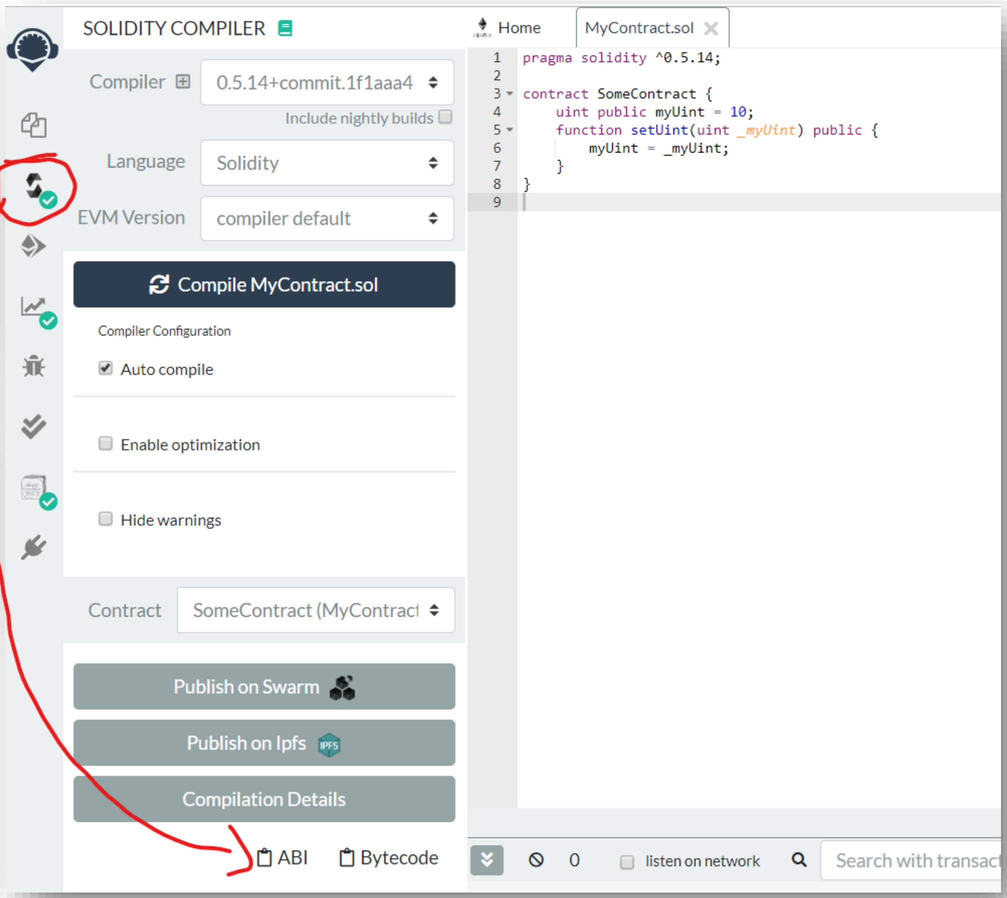


```
> var web3 = new Web3(Web3.providers.HttpProvider("http://localhost:7545"));
< undefined
> web3.eth.getAccounts().then(console.log);
< ▶ Promise {<pending>}
  (10) ["0xeeEa556133B5C76a2b92e8Df8F0Ff90BbbbD8ce6", "0xEddC05d9B2CB4D3965ee9d01C0c47960FAAbf273", "0xD152FdAea0F5c39bD
▶ 38Ce6c29312af14B93F40E3", "0xf2Af5a487b989EcFd4F6310E2A964c1649C0CD43", "0x4Fff1eD88e721140729fd3C179c807984bc70dfe",
  "0xc0BBc1Eb8F66b494B08Ec843e12742c351499fDa", "0x4AC266A48d587e541b6f6520246Ec1Fd94B9F6bd", "0x0e369aEDD00dC0bcc3fa3ff
  F63A46fA5fDD55A4c", "0x34C027923c8c3cE277E2aE8b0eadc4b7f486804c", "0xAeC5b09f722d36622F811631493E60f7ce6dB4c4"]
>
```

Update myUint using the ABI Array

Let’s do the same as we did before in the browser and see if it still works!

Copy the ABI Array from Remix:



Compiler: 0.5.14+commit.1f1aaa4

Language: Solidity

EVM Version: compiler default

[✓] Auto compile

[ ] Enable optimization

[ ] Hide warnings

Contract: SomeContract (MyContract

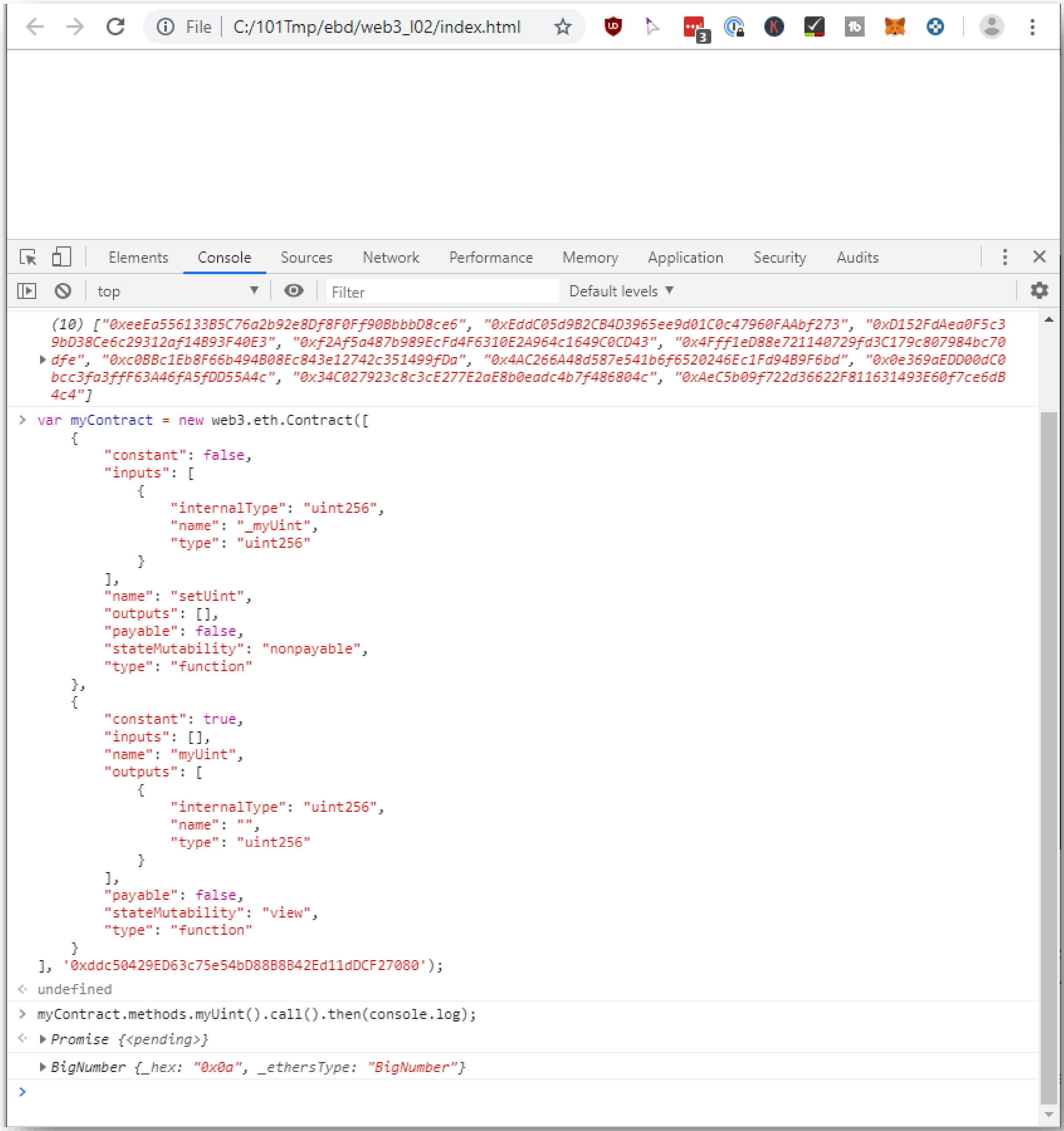
```
pragma solidity ^0.5.14;

contract SomeContract {
    uint public myUint = 10;
    function setUint(uint _myUint) public {
        myUint = _myUint;
    }
}
```

And enter the following code in node:

```
var myContract = new web3.eth.Contract(PASTE_ABI_ARRAY_HERE,
'CONTRACT_ADDRESS');
```

Then simply call via a very declarative function name:

```
myContract.methods.myUint().call().then(console.log).catch(console.error);
```



```
  (10) ["0xeeEa556133B5C76a2b92e8Df8F0Ff90BbbbD8ce6", "0xEddC05d9B2CB4D3965ee9d01C0c47960FAAbf273", "0xD152FdAea0F5c3
  9bD38Ce6c29312af14B93F40E3", "0xf2Af5a487b989EcFd4F6310E2A964c1649C0CD43", "0x4Fff1eD88e721140729fd3C179c807984bc70
▸ dfe", "0xc0BBc1Eb8F66b494B08Ec843e12742c351499fDa", "0x4AC266A48d587e541b6f6520246Ec1Fd94B9F6bd", "0x0e369aEDD00dC0
  bcc3fa3ffF63A46fA5fDD55A4c", "0x34C027923c8c3cE277E2aE8b0eadc4b7f486804c", "0xAeC5b09f722d36622F811631493E60f7ce6dB
  4c4"]
> var myContract = new web3.eth.Contract([
      {
          "constant": false,
          "inputs": [
              {
                  "internalType": "uint256",
                  "name": "_myUint",
                  "type": "uint256"
              }
          ],
          "name": "setUint",
          "outputs": [],
          "payable": false,
          "stateMutability": "nonpayable",
          "type": "function"
      },
      {
          "constant": true,
          "inputs": [],
          "name": "myUint",
          "outputs": [
              {
                  "internalType": "uint256",
                  "name": "",
                  "type": "uint256"
              }
          ],
          "payable": false,
          "stateMutability": "view",
          "type": "function"
      }
  ], '0xddc50429ED63c75e54bD88B8B42Ed11dDCF27080');
< undefined
> myContract.methods.myUint().call().then(console.log);
< ▸ Promise {<pending>}
  ▸ BigNumber {_hex: "0x0a", _ethersType: "BigNumber"}
>
```

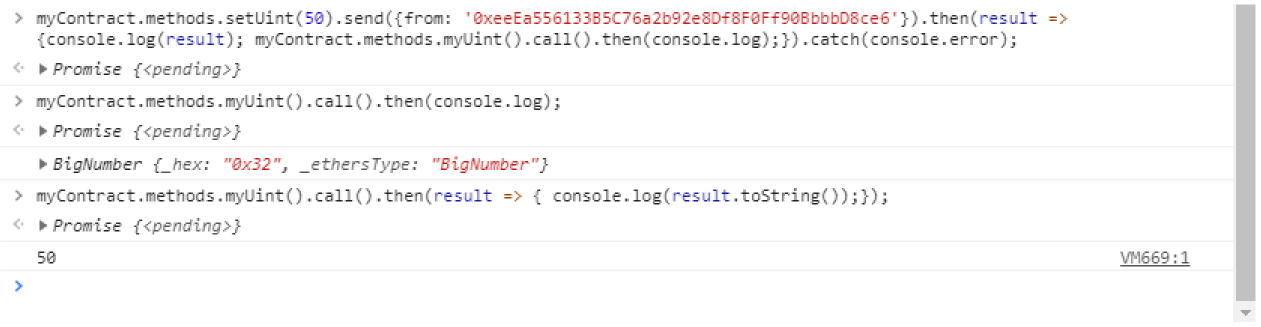
Now let's update the uint and call it again afterwards. But for this we must actually send off a transaction to our ganache. Web3 doesn't know which is the account we want to send the transaction from, so we have to set this first:

Then, once the transaction is mined (on Ganache instantaneously, but on a real network it might take 10-20 seconds), we read out the updated value.

```
myContract.methods.setUint(50).send({from:
'FIRST_ACCOUNT_FROM_GANACHE'}).then(result => {console.log(result);
myContract.methods.myUint().call().then(console.log);}).catch(console.error);
```

This should give you the hex-value of 0x32. You can also convert it to a regular string with:

```
myContract.methods.myUint().call().then(result => {
      console.log(result.toString());
});
```



```
> myContract.methods.setUint(50).send({from: '0xeeEa556133B5C76a2b92e8Df8F0Ff90BbbbD8ce6'}).then(result =>
  {console.log(result); myContract.methods.myUint().call().then(console.log);}).catch(console.error);
<· ▶ Promise {<pending>}
> myContract.methods.myUint().call().then(console.log);
<· ▶ Promise {<pending>}
   ▶ BigNumber {_hex: "0x32", _ethersType: "BigNumber"}
> myContract.methods.myUint().call().then(result => { console.log(result.toString());});
<· ▶ Promise {<pending>}
  50                                                                                          VM669:1
>
```

So, we did the same as before on the console with the browser. What does that mean? There is a standardized interface to “talk” to your smart contract.

Let’s play with this a bit further!

Congratulations, LAB is completed

From the Course “Ethereum Blockchain Developer – Build Projects in Solidity”

FULL COURSE:

https://www.udemy.com/course/blockchain-developer/?referralCode=E8611DF99D7E491DFD96