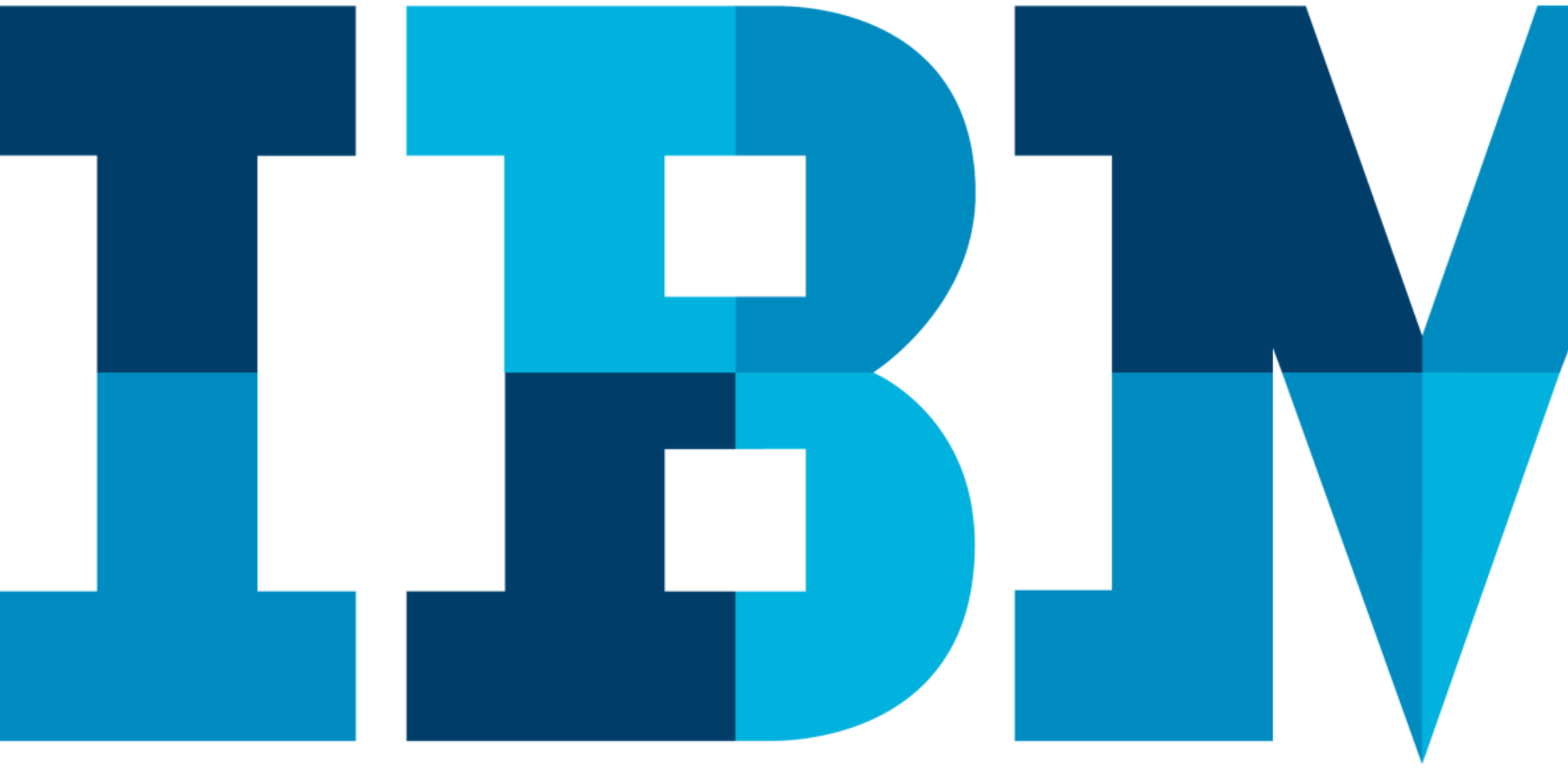
IBM Blockchain Platform Hands-On

Blockchain Technical Concepts



IBM

Table of Contents

Disclaimer

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated here.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts.
In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply.

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, ibm.com and [names of other referenced IBM products and services used in the presentation] are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

1 Preface

1.1 Overview

Underpinning blockchain is a set of well-known computer science concepts: linked lists, hash chains and the like. These form the basis of every blockchain in existence, and in order to fully grasp blockchain it is useful to be able to see these implemented from first principles.

1.2 Objectives

This short lab builds a complete JavaScript application to implement a very basic hash chain to demonstrate some of the concepts of blockchain.

It does not introduce any concepts or technology unique to a real-world blockchain implementation. Nor does it attempt to implement any of the advanced features of a blockchain, such as distributed peers, smart contracts or consensus.

1.3 Flow

We will start by implementing the basic *block* data structure, and linking instances of them together to form a block *chain*. We will look at how hash functions are used to provide continuity of the chain, and will test this out by simulating the tampering of data within the chain.

1.4 Tools

We will use Javascript as the language for implementing our hash chain, with Node as the run time environment and NPM as the package manager.

We will use an editor that can edit these Javascript files - VSCode is recommended, although if you wish to use an alternative editor, feel free to do so.

1.5 Prerequisites

Skill requirements:

- Basic knowledge of JavaScript is desirable.

Technical requirements (installed in the first section):

- Node, NPM and an editor for JavaScript files (e.g. VSCode).

2 The Hash Chain

2.1 Install prerequisites

Start here. Instructions are always shown on numbered lines like this one:

__ 1. Click the terminal window icon to bring up a new terminal window.

__ 2. Enter:

```
node -v
```

If you get a 'command not found' or similar error, you need to install Node from https://nodejs.org/en/download/.

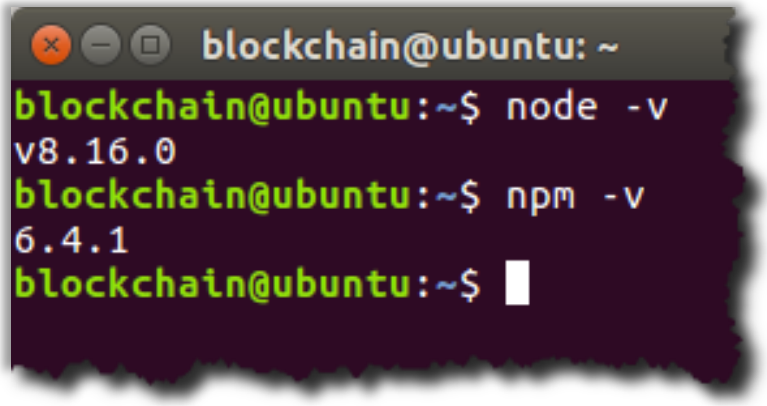
This lab was created using Node v8.16; other versions should work fine.

__ 3. Enter:

```
npm -v
```

NPM should have been installed as part of Node; however if you do get a 'command not found' or similar error, you need to install NPM from https://www.npmjs.com/get-npm.

This lab was created using NPM 6.4.1; other versions should work fine.



__ **4.** Ensure that you have a text editor available that can edit Javascript files. You don't need to start it yet.

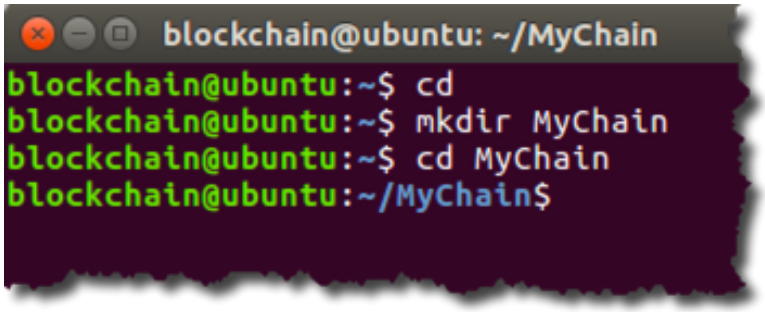
The VSCode icon looks like this:

If VSCode is not available, it can be downloaded from https://code.visualstudio.com/. Feel free to use an alternative Javascript code editor if you prefer.

2.2 Create the development environment

__ **5.** Create and navigate to a folder in which you can work; in the terminal window enter:

```
cd
mkdir MyChain
cd MyChain
```



This navigates to the user's home folder (not strictly necessary in a new terminal window, as that's the default), creates a new folder inside it that we will use to store the MyChain application, and tells the terminal window to navigate to the new folder. Subsequent commands in this window will run against files relative to this new folder.

In order to create the hashes for our blockchain, we will make use of a SHA256 library which is part of the Node crypto-js module. We must download this module and make it available to our application.

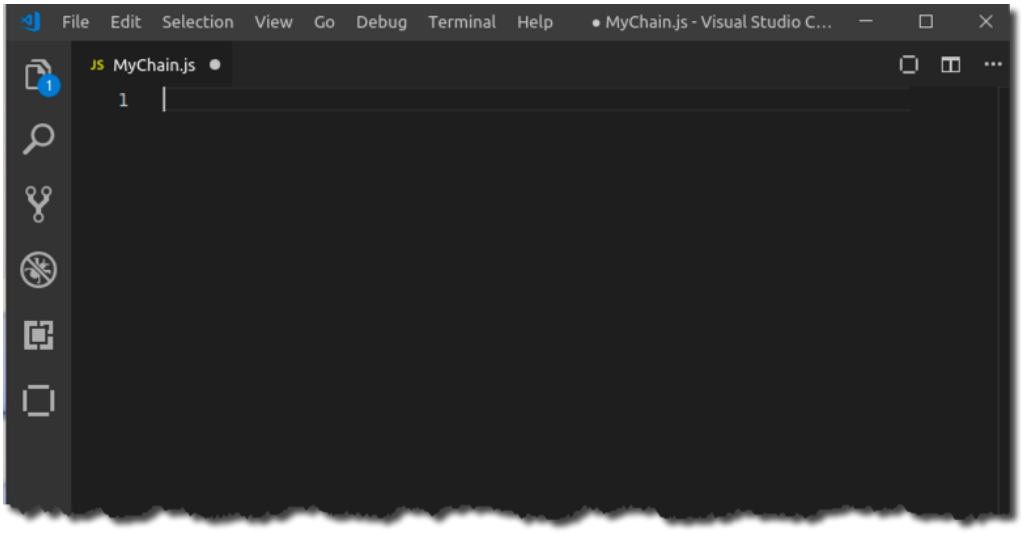
__ **6.** Enter:

```
npm install crypto-js
```

A node_modules folder will be created and the crypto-js library installed within it; warnings here can usually be ignored, as our application doesn't yet exist.

__ **7.** Start your editor and begin creating the blockchain application. If you installed VS Code for example, enter:

```
code MyChain.js
```



This will load VS Code and create a new Javascript file for you to edit.

> ***Note***
>
> Starting the VSCode editor for the first time might cause a web browser window to start. You can safely close this down.

2.3 Create a basic hash chain

In the JS file you created, we will now provide the basic implementation of a block.

__ **8.** Type (or copy/paste) the following into the MyChain.js editor window:

```
const SHA256 = require("crypto-js/sha256");

class Block {
    constructor(data, previousHash) {
        this.timestamp = Date.now();
        this.previousHash = previousHash;
        this.hash = this.getHash();
        this.data = data;
    }

    getHash() {
        return SHA256(this.previousHash + this.timestamp +
                      JSON.stringify(this.data)).toString();
    }
}
```

In this implementation, when a Block is instantiated you need to provide it some data; in a real blockchain this is usually a representation of a set of transactions. You also need to supply the hash of the previous block in the chain. It uses this to generate a hash of the block, together with the data and current timestamp. This helps ensure immutability of the chain.

__ **9.** Implement in the same file a Blockchain class definition:

```
class Blockchain {
    constructor() {
        this.blockchain = [new Block("Genesis Block", '')];
    }
    getLastBlock() {
        return this.blockchain[this.blockchain.length-1];
    }
    createBlock(data) {
        this.blockchain.push(new Block(data, this.getLastBlock().hash));
    }
}
```

The class represents the blockchain as an array of blocks, with a mandatory block zero which is called our genesis block. There is also a method to return the most recent block in the chain, and a factory for creating new blocks based on the supplied input data.

__ **10.** At the end of the file, implement code to create and test an instance of the blockchain.

```
myChain = new Blockchain();
console.log('\n-----\nNew blockchain created');
console.log(myChain);

myChain.createBlock("first set of transaction data");
console.log('\n-----\nAdded a block');
console.log(myChain);
myChain.createBlock("another set of transaction data");
console.log('\n-----\nAdded another block');
console.log(myChain);
```

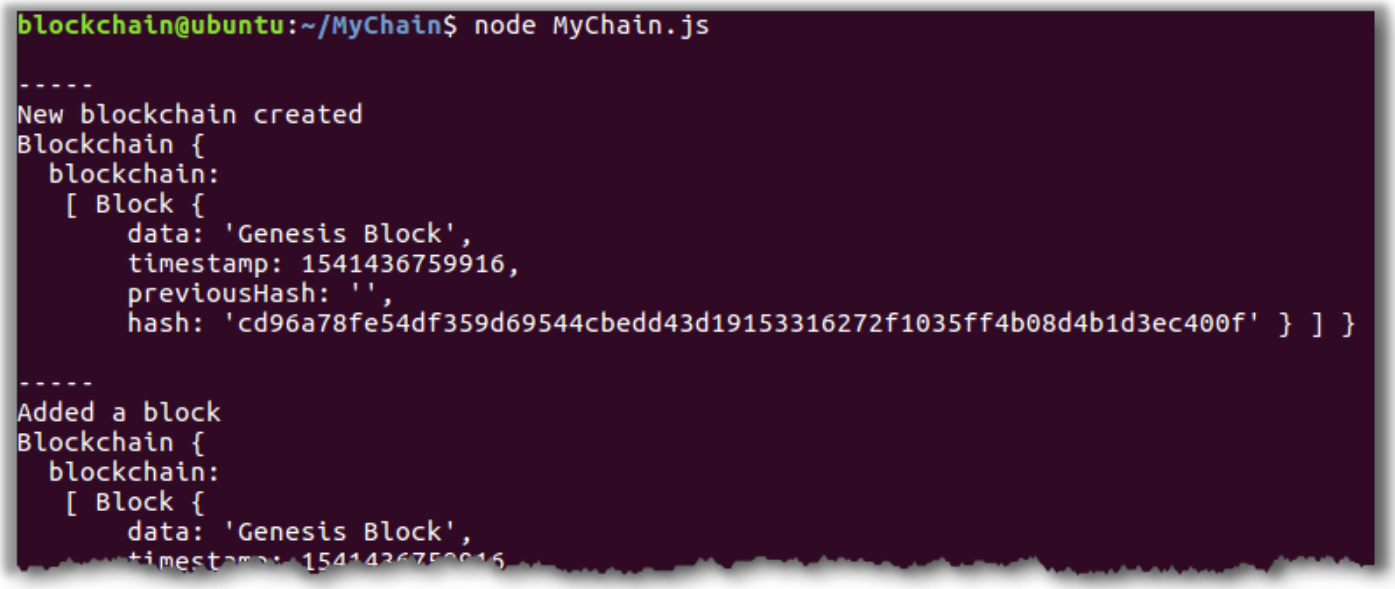
This instantiates a Blockchain and adds two blocks to it. At each stage, the contents of the blockchain are displayed on the console, to show how it grows.

__ **11.** Review your code and save it.

```
const SHA256 = require("crypto-js/sha256");

class Block {
  constructor(data, previousHash) {
    this.timestamp = Date.now();
    this.previousHash = previousHash;
    this.hash = this.getHash();
    this.data = data;
  }

  getHash() {
    return SHA256(this.previousHash + this.timestamp +
                  JSON.stringify(this.data)).toString();
  }
}

class Blockchain {
  constructor() {
    this.blockchain = [new Block("Genesis Block", '')];
  }

  getLastBlock() {
     return this.blockchain[this.blockchain.length-1];
  }

  createBlock(data) {
     this.blockchain.push(new Block(data, this.getLastBlock().hash));
  }
}

myChain = new Blockchain();
console.log('\n-----\nNew blockchain created');
console.log(myChain);
myChain.createBlock("first set of transaction data");
console.log('\n-----\nAdded a block');
console.log(myChain);
myChain.createBlock("another set of transaction data");
console.log('\n-----\nAdded another block');
console.log(myChain);
```

__ **12.** Return to your terminal window and run your Javascript application, enter:

```
node MyChain.js
```



__ **13.** Review the output from the program.

```
-----
New blockchain created
Blockchain {
  blockchain:
   [ Block {
       data: 'Genesis Block',
       timestamp: 1520596059987,
       previousHash: '',
       hash:
'df266d26ed337142cd00d128e090cdaf10a220a8e65ad6eca894577276c8e7fc' } ] }

-----
Added a block
Blockchain {
  blockchain:
   [ Block {
       data: 'Genesis Block',
       timestamp: 1520596059987,
       previousHash: '',
       hash:
'df266d26ed337142cd00d128e090cdaf10a220a8e65ad6eca894577276c8e7fc' },
     Block {
       data: 'first set of transaction data',
       timestamp: 1520596059993,
       previousHash:
'df266d26ed337142cd00d128e090cdaf10a220a8e65ad6eca894577276c8e7fc',
       hash:
'c02694901a64df111b5648da5181abfb65702ec193074bda4cd99af716596254' } ] }
```

```
-----
Added another block
Blockchain {
  blockchain:
   [ Block {
       data: 'Genesis Block',
       timestamp: 1520596059987,
       previousHash: '',
       hash:
'df266d26ed337142cd00d128e090cdaf10a220a8e65ad6eca894577276c8e7fc' },
     Block {
       data: 'first set of transaction data',
       timestamp: 1520596059993,
       previousHash:
'df266d26ed337142cd00d128e090cdaf10a220a8e65ad6eca894577276c8e7fc',
       hash:
'c02694901a64df111b5648da5181abfb65702ec193074bda4cd99af716596254' },
     Block {
       data: 'another set of transaction data',
       timestamp: 1520596059994,
       previousHash:
'c02694901a64df111b5648da5181abfb65702ec193074bda4cd99af716596254',
       hash:
'2be785a4020990626de8493ffd2c8dea7e346b06aec38665429e7430efa04791' } ] }
```

2.4 Help prevent modification of the hash chain

We will now implement code in the hash chain to check that the data stored within it has not been tampered. We will then test it by modifying the chain illegally.

__ **14.** Insert a new method into the Blockchain class to check the validity of the chain.

```
isBlockchainValid() {
    for (let i=1; i<this.blockchain.length; i++) {
        let currentBlock =  this.blockchain[i];
        let previousBlock = this.blockchain[i-1];
        if ((currentBlock.previousHash !== previousBlock.hash) ||
            (currentBlock.hash !== currentBlock.getHash())) {
            return false;
        }
    }
    return true;
```

```
    }
```

This method ensures that the hash of each block is correctly refers to the hash recorded in the previous block, and that the contents of the block have been hashed correctly. This helps prevent the data in a recorded block from being modified without rendering the validity check invalid.

__ **15.** At the end of the entire program, add code to call the new method, modify a block 'illegally' and then call the method again.

```
console.log('Is the chain valid: ' + myChain.isBlockchainValid());
myChain.blockchain[1].data = "changed set of transactions";
console.log('Modified transaction data in a block');
console.log('Is the chain valid: ' + myChain.isBlockchainValid());
```

__ **16.** Review your code and save it.

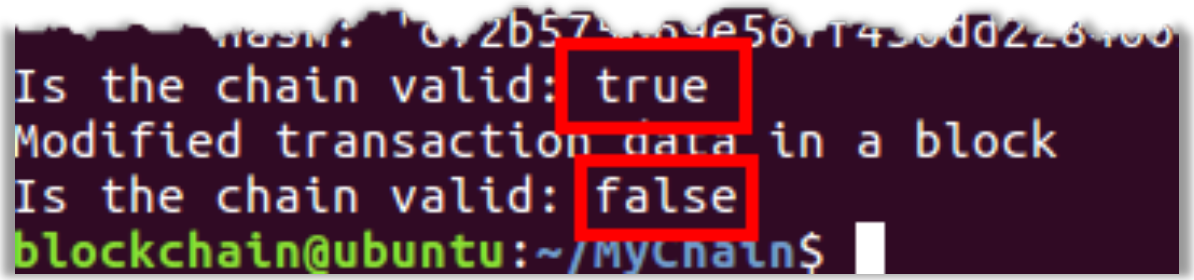
```
1.  const SHA256 = require("crypto-js/sha256");
2.
3.  class Block {
4.    constructor(data, previousHash) {
5.      this.timestamp = Date.now();
6.      this.previousHash = previousHash;
7.      this.hash = this.getHash();
8.      this.data = data;
9.    }
10.
11.   getHash() {
12.     return SHA256(this.previousHash + this.timestamp +
13.                   JSON.stringify(this.data)).toString();
14.   }
15. }
16.
17. class Blockchain {
18.   constructor() {
19.     this.blockchain = [new Block("Genesis Block", '')];
20.   }
21.
22.   getLastBlock() {
23.      return this.blockchain[this.blockchain.length-1];
24.   }
25.
26.   createBlock(data) {
27.      this.blockchain.push(new Block(data, this.getLastBlock().hash));
28.   }
29.
30.   isBlockchainValid() {
31.     for (let i=1; i<this.blockchain.length; i++) {
32.       let currentBlock = this.blockchain[i];
```

```
33.       let previousBlock = this.blockchain[i-1];
34.        if ((currentBlock.previousHash !== previousBlock.hash) ||
35.            (currentBlock.hash !== currentBlock.getHash())) {
36.            return false;
37.        }
38.     }
39.     return true;
40.   }
41.
42. }
43.
44. myChain = new Blockchain();
45. console.log('\n-----\nNew blockchain created');
46. console.log(myChain);
47. myChain.createBlock("first set of transaction data");
48. console.log('\n-----\nAdded a block');
49. console.log(myChain);
50. myChain.createBlock("another set of transaction data");
51. console.log('\n-----\nAdded another block');
52. console.log(myChain);
53.
54. console.log('Is the chain valid: ' + myChain.isBlockchainValid());
55. myChain.blockchain[1].data = "changed set of transactions";
56. console.log('Modified transaction data in a block');
57. console.log('Is the chain valid: ' + myChain.isBlockchainValid());
```

__ **17.** Back in the terminal window, run the modified program.

```
node MyChain.js
```

__ **18.** Review the output.



Notice that once the chain has been illegally modified, it is no longer valid.
This is only a very basic implementation of a single-process centralised hashchain.

Real-world blockchains are significantly more complicated than this: they are distributed processing systems where different network participants use a process of consensus to agree on the contents of a block. The underlying data structure is based on the fundamental principles shown here however, which help prove immutability of the data stored within the chain.

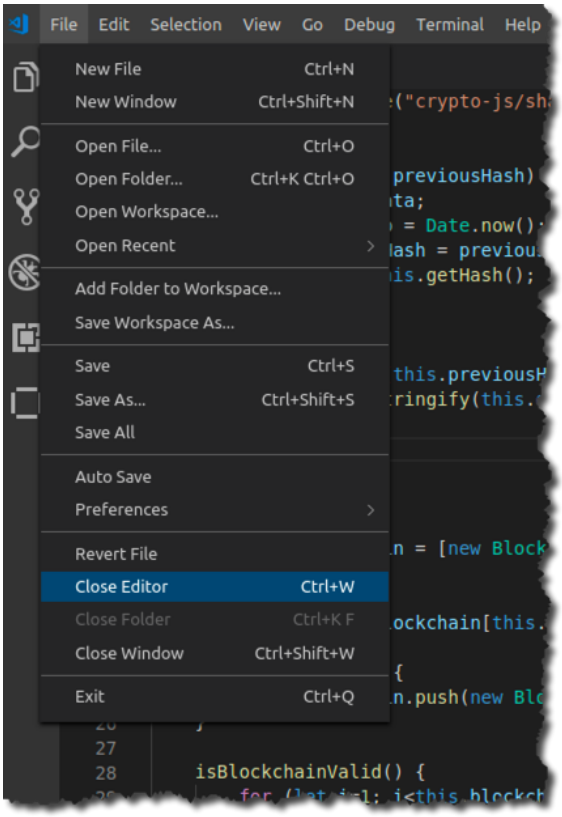
Congratulations! You have completed this lab.

3 Resetting the lab environment

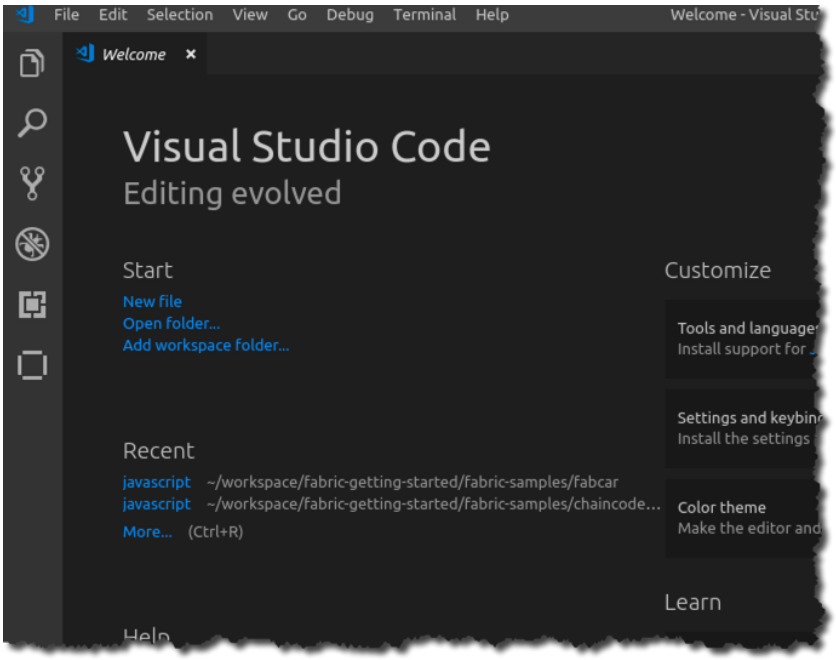
In order to reset the lab environment (for example, to do an additional lab or to restart this one) then you should reset VSCode and remove the files you downloaded or created from the disk. This section explains how to do this.

3.1 Resetting VSCode

__ **19.** In VSCode, with the MyChain.js file loaded click File -> Close Editor.



__ **20.** To return to the initial VSCode screen click Help -> Welcome.



__ **21.** Exit VSCode by clicking File -> Exit.

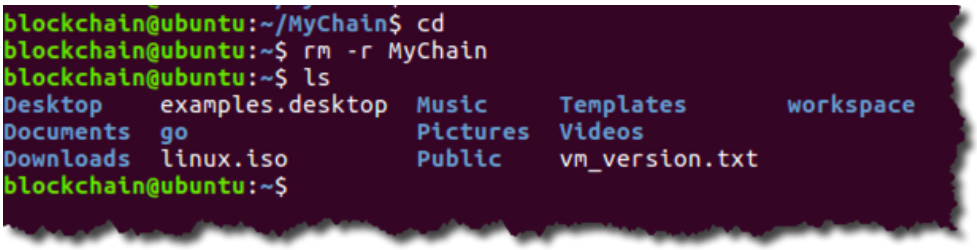
3.2 Resetting the file system

The crypto-js library files that you downloaded and installed in the first section is stored along with your source code in the MyChain folder. To reset the file system you simply need to delete this folder.

__ **22.** In the terminal window, enter the commands:

```
cd
rm -r MyChain
```

You can then verify that the MyChain folder no longer exists by using the **ls** command; MyChain will not be listed.

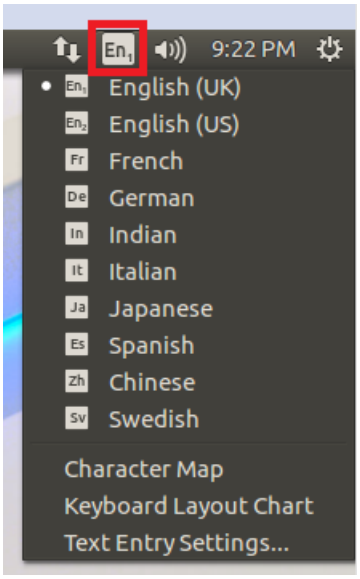


__ **23.** Close the terminal window.

Appendix A. Changing the keyboard language

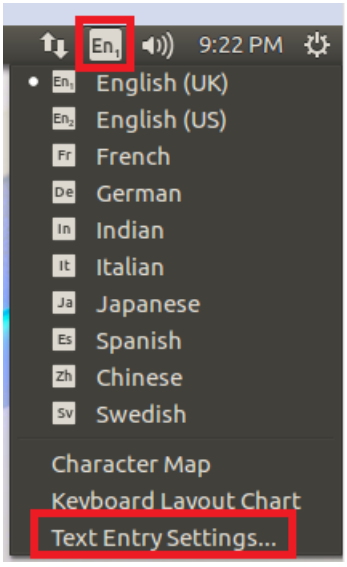
To change the keyboard language to enable you to use laptops with keyboards from different countries follow these steps:

__ **1.** Click on the  icon in the top right and look at the list of languages shown:

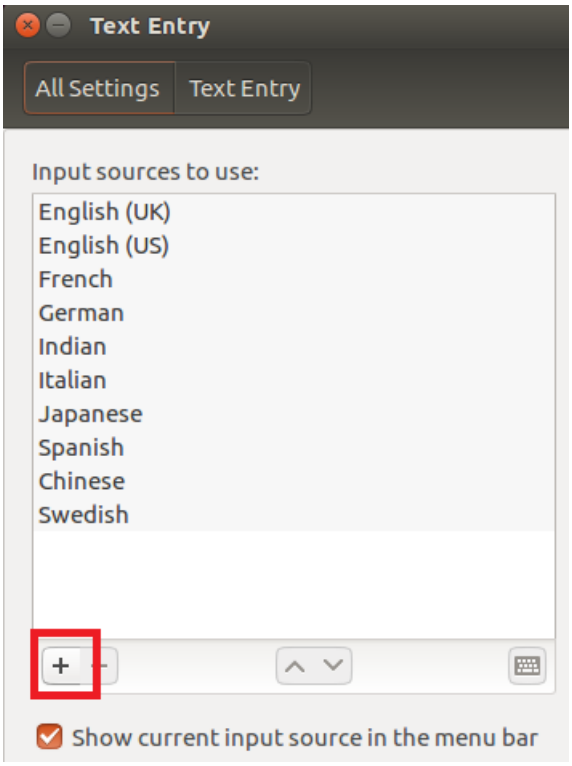


If your required language is present, simply select your language of choice and you are done; skip the following steps.

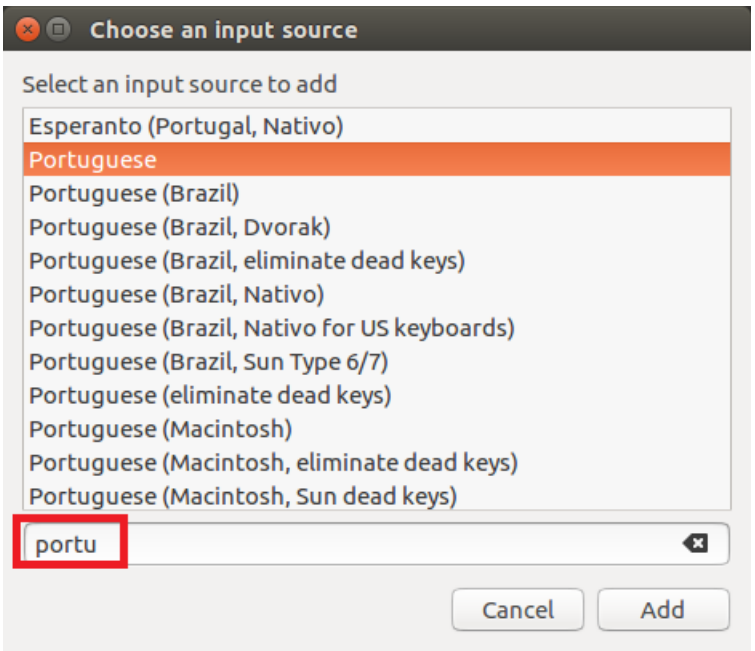
__ **2.** If your required language is not present, select **Text Entry Settings...** from the list:



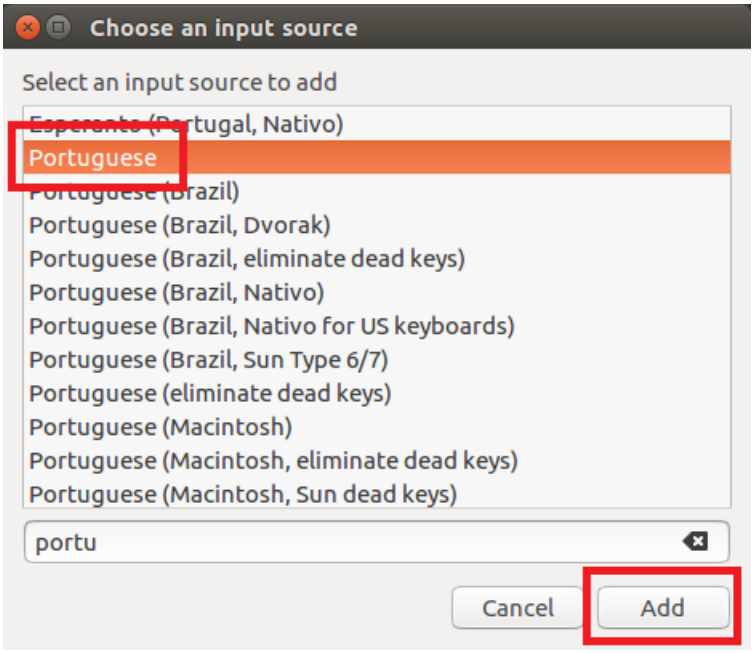
__ 3. **Select** the + symbol to add a new language:



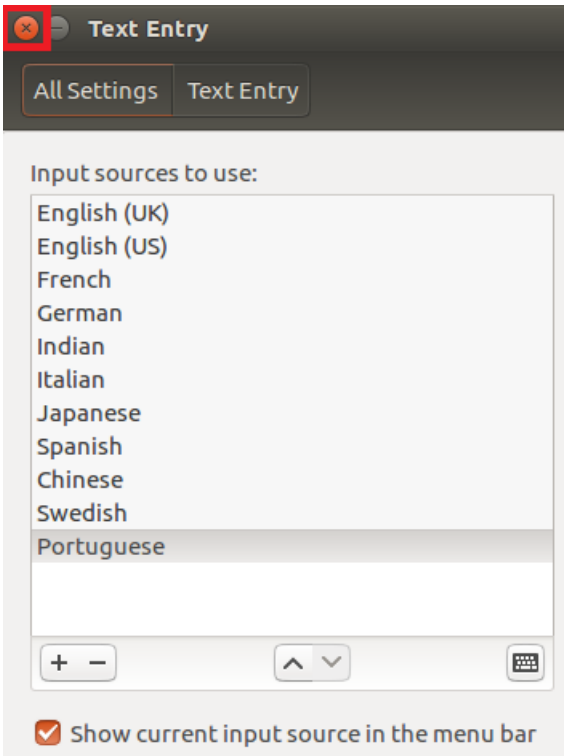
__ 4. Start to type your language until you see your language's name appear – we will add Portuguese as an example here:



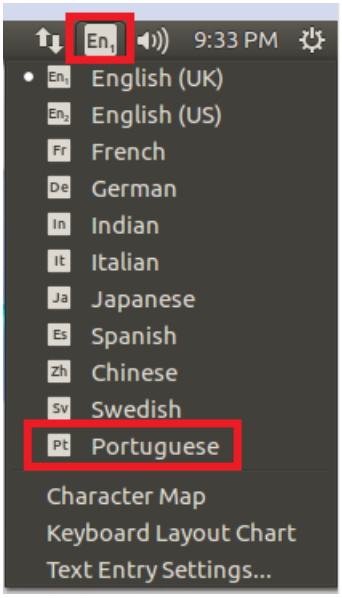
__ **5.** Select your chosen keyboard and click '**Add**':



__ **6.** **Close** the Settings box by clicking on the "**x**" in the top-right of the dialogue:



__ **7.** Select the  in the top right of the screen and select your new keyboard:



__ **8.** Your keyboard is now ready to use.

We Value Your Feedback!

- Please ask your instructor for an evaluation form. Your feedback is very important to us as we use it to continually improve the lab material.
- If no forms are available, or you want to give us extra information after the lab has finished, please send your comments and feedback to "**blockchain@uk.ibm.com**"