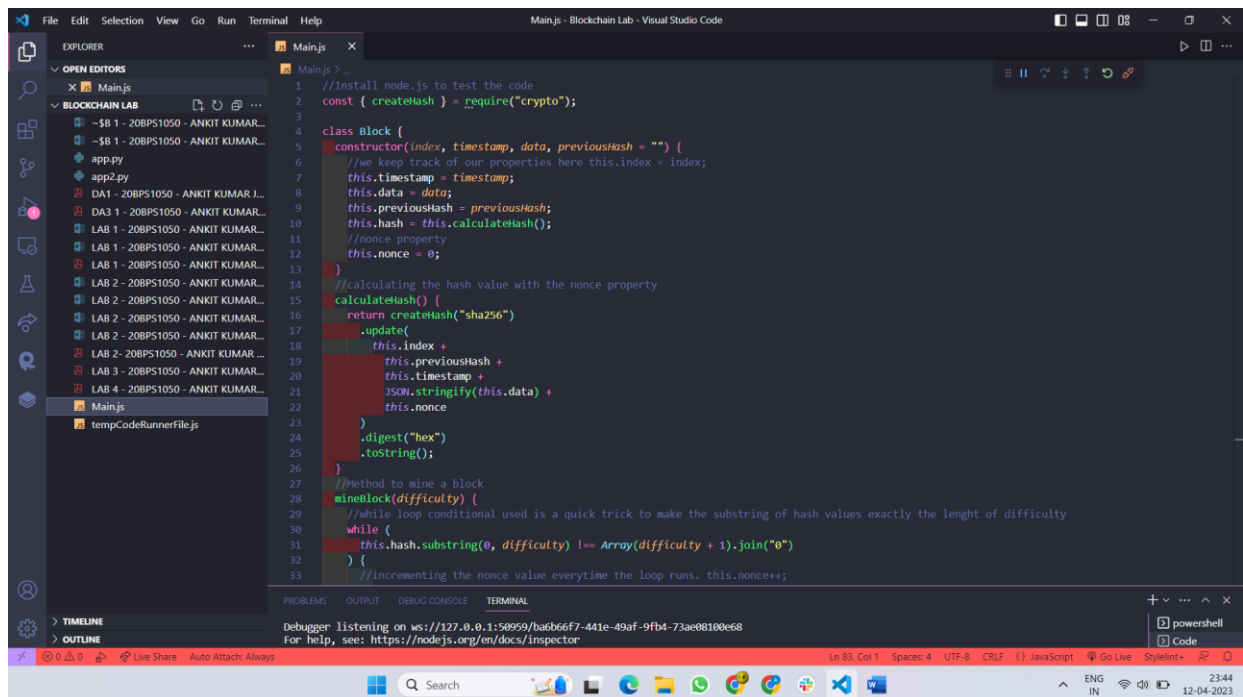


NAME: ANKIT KUMAR JHA  
REG.NO.20BPS1050

## EXERCISE 6

### IMPLEMENT HASH PUZZLE



```
1 //Install node.js to test the code
2 const { createHash } = require("crypto");
3
4 class Block {
5   constructor(index, timestamp, data, previousHash = "") {
6     //we keep track of our properties here this.index = index;
7     this.timestamp = timestamp;
8     this.data = data;
9     this.previousHash = previousHash;
10    this.hash = this.calculateHash();
11    //nonce property
12    this.nonce = 0;
13  }
14  //calculating the hash value with the nonce property
15  calculateHash() {
16    return createHash("sha256")
17      .update(
18        this.index +
19        this.previousHash +
20        this.timestamp +
21        JSON.stringify(this.data) +
22        this.nonce
23      )
24      .digest("hex")
25      .toString();
26  }
27  //Method to mine a block
28  mineBlock(difficulty) {
29    //while loop conditional used is a quick trick to make the substring of hash values exactly the length of difficulty
30    while (
31      this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0")
32    ) {
33      //incrementing the nonce value everytime the loop runs. this.nonce++;
34    }
35  }
36}
```

main.js

```
//Install node.js to test the code
const { createHash } = require("crypto");

class Block {
  constructor(index, timestamp, data, previousHash = "") {
    //we keep track of our properties here this.index = index;
    this.timestamp = timestamp;
    this.data = data;
    this.previousHash = previousHash;
    this.hash = this.calculateHash();
    //nonce property
    this.nonce = 0;
  }
  //calculating the hash value with the nonce property
  calculateHash() {
```

```

return createHash("sha256")
    .update(
        this.index +
        this.previousHash +
        this.timestamp +
        JSON.stringify(this.data) +
        this.nonce
    )
    .digest("hex")
    .toString();
}
//Method to mine a block
mineBlock(difficulty) {
    //while loop conditional used is a quick trick to make the substring of hash values exactly the
    //length of difficulty
    while (
        this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0")
    ) {
        //incrementing the nonce value everytime the loop runs. this.nonce++;
        //recalculating the hash value
        this.hash = this.calculateHash();
    }
    //logging when a block is created
    console.log("Block mined: " + this.hash);
}
}
class Blockchain {
    constructor() {
        this.chain = [this.createGenesisBlock()];
        //adding a difficulty property to the Blockchain class
        this.difficulty = 4;
    }
    createGenesisBlock() {
        return new Block(0, "02/01/2018", "Genesis Block", "0");
    }
    getlatestBlock() {
        return this.chain[this.chain.length - 1];
    }
    addBlock(newBlock) {
        newBlock.previousHash = this.getlatestBlock().hash;
        //We commented the earlier method that adds a block directly //newBlock.hash =
        newBlock.calculateHash();
    }
}

```

```
//New method to mine the block //Customizable difficulty value

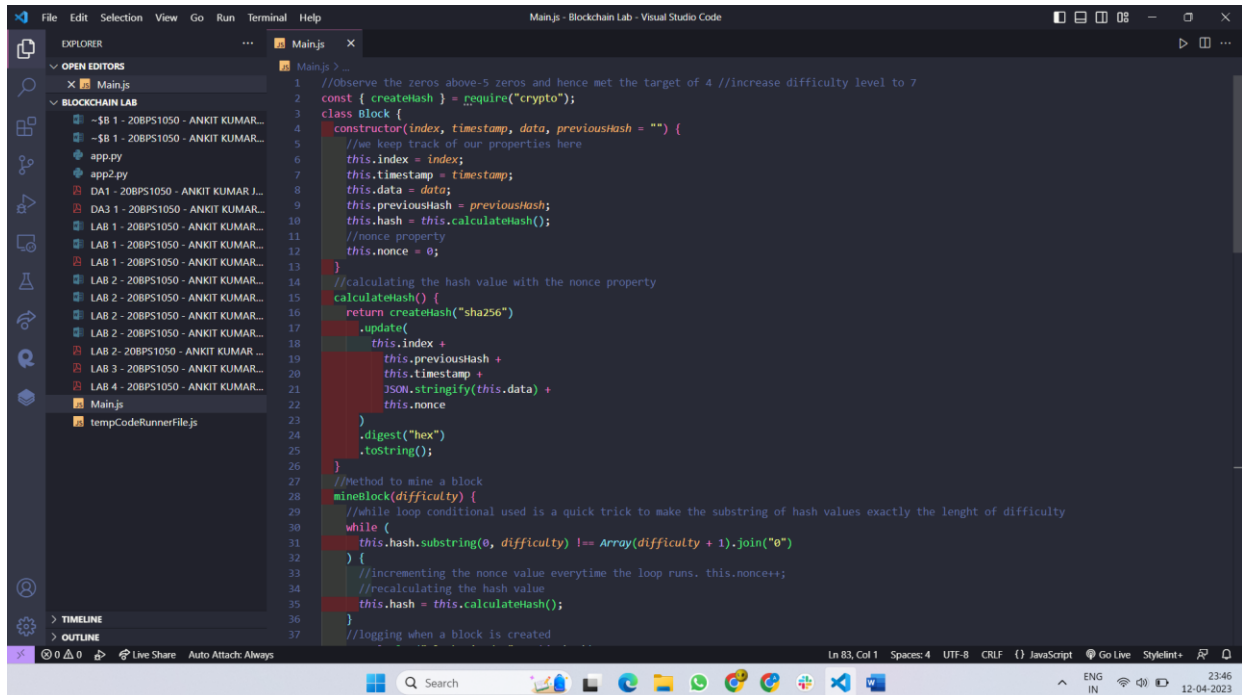
newBlock.mineBlock(this.difficulty);
this.chain.push(newBlock);
}
isChainValid() {
  for (let i = 1; i < this.chain.length; i++) {
    const currentBlock = this.chain[i];
    const previousBlock = this.chain[i - 1];
    if (currentBlock.hash !== currentBlock.calculateHash()) {
      return false;
    } //check for hash calculations
    if (currentBlock.previousHash !== previousBlock.hash) {
      return false;
    } //check whether current block points to the correct previous block
  }
  return true;
}
}
let koreCoin = new Blockchain();
console.log("Mining block 1...");
koreCoin.addBlock(new Block(1, "01/01/2018", { amount: 20 }));
console.log("Mining block 2...");

koreCoin.addBlock(new Block(2, "02/01/2018", { amount: 40 }));
console.log("Mining block 3...");
koreCoin.addBlock(new Block(3, "02/01/2018", { amount: 40 }));
```

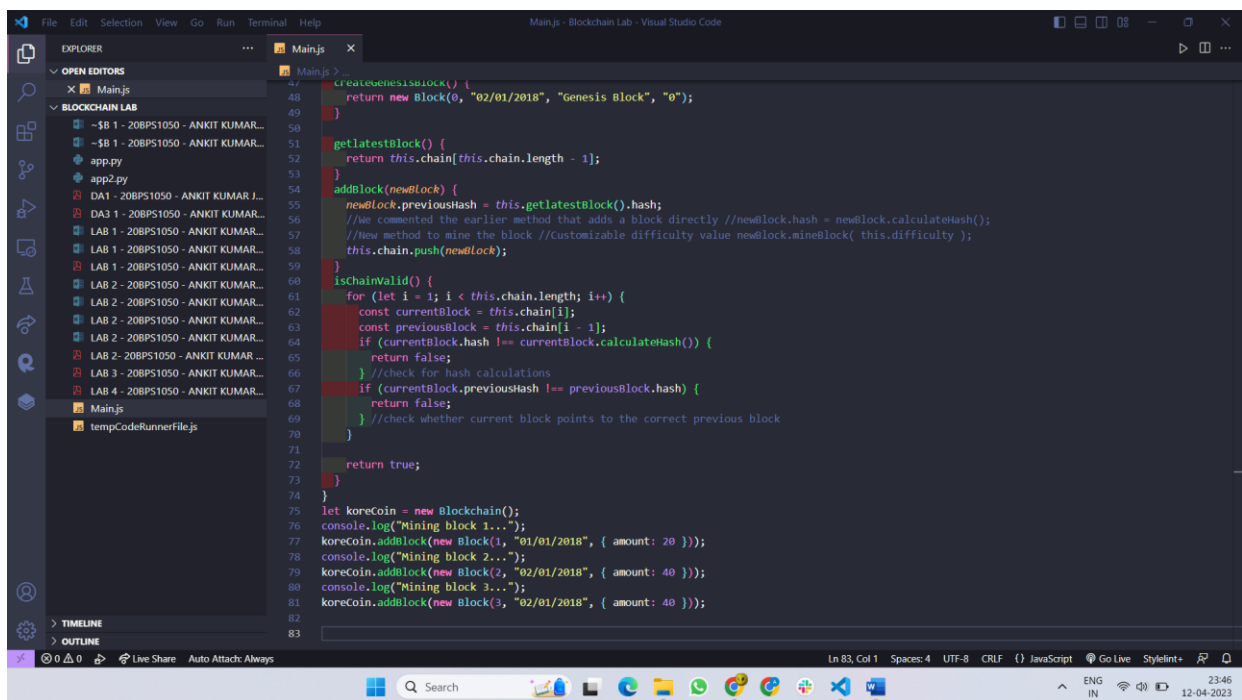
## Output-

```
Mining block 1...
Block mined: 0000023168f87d968813b22c4dc92f60c127ff5084af8487d913d497ea7a7900
Mining block 2...
Block mined: 00008e0c291aaf728e015855328b14e651231cce209e6413503fd299e0df6c5e
Mining block 3...
Block mined: 0000fb4a126ef4c1c3c93bf2ed25e8db4c7da2ec89a46aad4f7bf092afd8b6b4
```

# Changing difficulty to 7



```
1 //Observe the zeros above-5 zeros and hence met the target of 4 //increase difficulty level to 7
2 const { createHash } = require("crypto");
3 class Block {
4   constructor(index, timestamp, data, previousHash = "") {
5     //we keep track of our properties here
6     this.index = index;
7     this.timestamp = timestamp;
8     this.data = data;
9     this.previousHash = previousHash;
10    this.hash = this.calculateHash();
11    //nonce property
12    this.nonce = 0;
13  }
14  //calculating the hash value with the nonce property
15  calculateHash() {
16    return createHash("sha256")
17      .update(
18        this.index +
19        this.previousHash +
20        this.timestamp +
21        JSON.stringify(this.data) +
22        this.nonce
23      )
24      .digest("hex")
25      .toString();
26  }
27  //method to mine a block
28  mineBlock(difficulty) {
29    //while loop conditional used is a quick trick to make the substring of hash values exactly the lenght of difficulty
30    while (
31      this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0")
32    ) {
33      //incrementing the nonce value everytime the loop runs. this.nonce++;
34      //recalculating the hash value
35      this.hash = this.calculateHash();
36    }
37    //logging when a block is created
```



```
47 createGenesisBlock() {
48   return new Block(0, "02/01/2018", "Genesis Block", "0");
49 }
50
51 getLatestBlock() {
52   return this.chain[this.chain.length - 1];
53 }
54 addBlock(newBlock) {
55   newBlock.previousHash = this.getLatestBlock().hash;
56   //we commented the earlier method that adds a block directly //newBlock.hash = newBlock.calculateHash();
57   //New method to mine the block //Customizable difficulty value newBlock.mineBlock( this.difficulty );
58   this.chain.push(newBlock);
59 }
60 isValidChain() {
61   for (let i = 1; i < this.chain.length; i++) {
62     const currentBlock = this.chain[i];
63     const previousBlock = this.chain[i - 1];
64     if (currentBlock.hash !== currentBlock.calculateHash()) {
65       return false;
66     } //check for hash calculations
67     if (currentBlock.previousHash !== previousBlock.hash) {
68       return false;
69     } //check whether current block points to the correct previous block
70   }
71   return true;
72 }
73
74 let koreCoin = new Blockchain();
75 console.log("Mining block 1...");
76 koreCoin.addBlock(new Block(1, "01/01/2018", { amount: 20 }));
77 console.log("Mining block 2...");
78 koreCoin.addBlock(new Block(2, "02/01/2018", { amount: 40 }));
79 console.log("Mining block 3...");
80 koreCoin.addBlock(new Block(3, "02/01/2018", { amount: 40 }));
81
82
83
```

## main.js

```
//Observe the zeros above-5 zeros and hence met the target of 4 //increase difficulty level to 7
const { createHash } = require("crypto");
class Block {
  constructor(index, timestamp, data, previousHash = "") {
    //we keep track of our properties here
    this.index = index;
    this.timestamp = timestamp;
    this.data = data;
    this.previousHash = previousHash;
    this.hash = this.calculateHash();
    //nonce property
    this.nonce = 0;
  }
  //calculating the hash value with the nonce property
  calculateHash() {
    return createHash("sha256")
      .update(
        this.index +
        this.previousHash +
        this.timestamp +
        JSON.stringify(this.data) +
        this.nonce
      )
      .digest("hex")
      .toString();
  }
  //Method to mine a block
  mineBlock(difficulty) {
    //while loop conditional used is a quick trick to make the substring of hash values exactly the
    //length of difficulty
    while (
      this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0")
    ) {
      //incrementing the nonce value everytime the loop runs. this.nonce++;
      //recalculating the hash value
      this.hash = this.calculateHash();
    }
    //logging when a block is created
    console.log("Block mined: " + this.hash);
  }
}
```

```

}
class Blockchain {

constructor() {
  this.chain = [this.createGenesisBlock()];
  //adding a difficulty property to the Blockchain class
  this.difficulty = 7;
}

createGenesisBlock() {
  return new Block(0, "02/01/2018", "Genesis Block", "0");
}

getlatestBlock() {
  return this.chain[this.chain.length - 1];
}

addBlock(newBlock) {
  newBlock.previousHash = this.getlatestBlock().hash;
  //We commented the earlier method that adds a block directly //newBlock.hash =
newBlock.calculateHash();
  //New method to mine the block //Customizable difficulty value newBlock.mineBlock(
this.difficulty );
  this.chain.push(newBlock);
}

isChainValid() {
  for (let i = 1; i < this.chain.length; i++) {
    const currentBlock = this.chain[i];
    const previousBlock = this.chain[i - 1];
    if (currentBlock.hash !== currentBlock.calculateHash()) {
      return false;
    } //check for hash calculations
    if (currentBlock.previousHash !== previousBlock.hash) {
      return false;
    } //check whether current block points to the correct previous block
  }

  return true;
}
}

let koreCoin = new Blockchain();
console.log("Mining block 1...");
koreCoin.addBlock(new Block(1, "01/01/2018", { amount: 20 }));
console.log("Mining block 2...");

```

```
koreCoin.addBlock(new Block(2, "02/01/2018", { amount: 40 }));
```

```
console.log("Mining block 3...");
```

```
koreCoin.addBlock(new Block(3, "02/01/2018", { amount: 40 }));
```

## Output –

```
Mining block 1...  
Block mined: 0000000099da284da2a3a454e75576632ab139495dfd9cb160080e534bbf90d5  
Mining block 2...  
Block mined: 00000007f6fb7d91f0a6ee2472aa6ab5d7fe73e04b0d2e5cfc6feca75c571cb2  
Mining block 3...  
Block mined: 000000086f9f621bd96e61faae30fb3f8f4da1e992f50a76d7c2d658d4d49ac6
```