**Blockchain with Python**

**By**

**Redzepi Enis**

**&**

**Bedjeti Bedjet**

**Course:**

**Programming with Advanced Computer Languages**

**16.12.2019**

**Dr. Mario Silic**

**1. What is a Blockchain**

In order to get an understanding of what a Blockchain is, please refer to the following link: http://graphics.reuters.com/TECHNOLOGY-BLOCKCHAIN/010070P11GN/index.html

Short video introduction: https://www.youtube.com/watch?v=r43LhSUUGTQ

**2. The Code**

Right below you can see our simple Code for a simple Blockchain (extracted locally from JetBrains PyCharm). The explanations are to be found within the Code (after every #). The code of a blockchain can of course go much more in depth, but the main aspects are to be found in our code (namely the definition of a single block and of a whole blockchain). At page 3, three different yet simple uses of the code are described (Addition of transactions, Balance output, Validation). We also added some lines of code at the end, in order to try hosting the blockchain on a webpage (see page 3-4).

The code can also be found on GitHub (along with these word file): https://github.com/enredzepi/School\_project.git

#Several imports needed for the code  
#If you cannot import them, then you'll  
#probably have to install certain modules  
import hashlib  
import json  
from datetime import datetime  
from time import time  
from uuid import uuid4  
from flask import Flask  
from flask import jsonify  
from pprint import pprint  
  
#Basically what we have to do for a Blockchain to work  
#is first of all defining what a single Block is  
#and then defining how the Chain is made up of these  
#single Blocks we already defined  
  
#Basically, what we have to do for a blockchain to work is to define what a SINGLE BLOCK looks like  
#So we defined the timestamp, the transaction and the hashes of a block  
#To calculate our hashes we imported the libraries hashlib and json  
#Additionally, our blockchain is mineable (number of “0” in the beginning of the hash value)  
#So we also defined an algorithm and the difficulty of the mining process.  
class Block():  
 def \_\_init\_\_(self, nonce, tstamp, transactionsList, prevhash='', hash=''):  
 self.nonce = nonce  
 self.tstamp = tstamp  
 self.transactionsList = transactionsList # list of dictionary values  
 self.prevhash = prevhash  
 if hash == '':  
 self.hash = self.calcHash()  
 else:  
 self.hash = hash  
  
 def calcHash(self):  
 block\_string = json.dumps(  
 {"nonce": self.nonce, "tstamp": str(self.tstamp), "transactions": self.transactionsList, "prevhash": self.prevhash}, sort\_keys=True).encode()  
 return hashlib.sha256(block\_string).hexdigest()  
  
 def mineBlock(self, diffic):  
 while (self.hash[:diffic] != "0" \* diffic):  
 self.nonce += 1  
 self.hash = self.calcHash()  
  
 def toDict(self):  
 return {"nonce": self.nonce, "tstamp": str(self.tstamp), "transactionsList": self.transactionsList, "prevhash": self.prevhash, "hash": self.hash}  
  
#Now that we have defined our single blocks we can start chaining our blocks to a blockchain.  
#So we defined the class BlockChain. To get started we need a first block.  
#To do this we use the generate function to get the initial block of the chain.  
#With the definition of getLastBlock and minePendingTransaction we define the (previous) and further blocks of our blockchain  
#and we also define to which address we want to pay the reward for the newly mined block.  
#At the end we defined also a validity check to make sure that the blockchain is integer  
#by checking the hashes of our blockchain.  
#By doing this, we can show if someone changed or tried to change the blockchain.  
class BlockChain():  
 def \_\_init\_\_(self):  
 self.chain = []  
 self.pendingTransactions = []  
 self.mining\_reward = 50  
 self.difficulty = 3  
 self.generateGenesisBlock()  
  
 def generateGenesisBlock(self):  
 dect = {"nonce": 0, "tstamp": '01/01/2019',  
 "transactionsList": [{"from\_address": None, "to\_address": None, "amount": 0}, ], "hash": ''}  
 b = Block(\*\*dect)  
 self.chain.append(b.toDict())  
  
 def getLastBlock(self):  
 return Block(\*\*self.chain[-1])  
  
 def minePendingTransaction(self, mining\_reward\_address):  
 block = Block(0, str(datetime.now()), self.pendingTransactions)  
 block.prevhash = self.getLastBlock().hash  
 block.mineBlock(self.difficulty)  
 print("Block is mined to got reward", self.mining\_reward)  
 self.chain.append(block.toDict())  
 self.pendingTransactions = [  
 {"from\_address": None, "to\_address": mining\_reward\_address, "amount": self.mining\_reward}, ]  
  
 def createTransaction(self, from\_address, to\_address, amount):  
 self.pendingTransactions.append({  
 'from\_address': from\_address,  
 'to\_address': to\_address,  
 'amount': amount,  
 })  
  
 def isChainValid(self):  
 for index in range(1, len(self.chain)):  
 currb = Block(\*\*self.chain[index])  
 prevb = Block(\*\*self.chain[index - 1])  
 if currb.hash != currb.calcHash():  
 return False  
 if currb.prevhash != prevb.hash:  
 return False  
 return True

def calcBalance(self, address):  
 balance = 0  
 for index in range(len(self.chain)):  
 dicList = self.chain[index]["transactionsList"]  
 for dic in dicList:  
 if dic["to\_address"] == address:  
 balance += dic["amount"]  
 if dic["from\_address"] == address:  
 balance -= dic["amount"]  
 return balance  
  
#Creating our Blockchain under the name "our\_bchain"  
our\_bchain = BlockChain()

#-----------------------------------------------------------------------------------  
# 1.USE: Manually adding transactions to the Blockchain, going from one User (Enis) to another (Bedjet)  
our\_bchain.createTransaction('Enis', 'Bedjet', 1050)  
our\_bchain.createTransaction('Bedjet', 'Enis', 300)  
#Enis is mining it, so he will get a reward from the system  
our\_bchain.minePendingTransaction('Enis')  
our\_bchain.createTransaction('Enis', 'Bedjet', 300)  
our\_bchain.createTransaction('Bedjet', 'Enis', 800)  
#Now Bedjet is going to mine, but the reward will not yet be shown  
#in his balance: #The reward will be added after the current pending  
#Transaction is over, i.e after a subsequent Transaction is succesfully mined  
our\_bchain.minePendingTransaction('Bedjet')  
  
# 2.USE: Having a look at the balance (of a specific User)  
print('Bedjet, your balance is: ', our\_bchain.calcBalance('Bedjet'))  
print('Enis, your balance is: ', our\_bchain.calcBalance('Enis'))  
  
# 3.USE: Validation of Blocks and Chain; Testing whether it was hacked or not  
if (our\_bchain.isChainValid()):  
 print('valid blockhain')  
else:  
 print('hacked blockchain')  
#------------------------------------------------------------------------------------  
  
#Hosting the whole thing on a webpage  
app=Flask(\_\_name\_\_)  
  
node\_id=str(uuid4()).replace('-','')  
  
#By clicking on the link that gets put out when running the code,  
#you will get on the main page:  
@app.route("/")  
def hello():  
 return "Hello, you are in the main page of the node"  
  
#You can add at the end of the link "/mine" to get on the particular  
#page defined below:  
@app.route("/mine",methods=['GET'])  
def mine():  
 return "we'll mine a new block"  
  
@app.route('/transactions/new',methods=['POST'])  
def new\_transaction():  
 return "we'll add a new transaction here"  
  
#If you add at the end of the link "/chain" you can have  
#a look at the chain: You'll see the different hash codes  
#already generated in the chain, when a particular  
#transaction was done etc.:  
@app.route('/chain',methods=['GET'])  
def full\_chain():  
 response = {  
 'chain':our\_bchain.chain,  
 'length':len(our\_bchain.chain),  
 }  
 return jsonify(response), 200  
  
if \_\_name\_\_=="\_\_main\_\_":  
 app.run()