**PART I – Blockchain Accounting Models**

**1.1 UTXO Exercise**

Fetch the UTXOs of 1Dorian4RoXcnBv9hnQ4Y2C1an6NJ4UrjX. What is the total balance of spendable Satoshis?

**Python script**

import json

import requests

# Target address

address = "1Dorian4RoXcnBv9hnQ4Y2C1an6NJ4UrjX"

# Fetch UTXOs

resp = requests.get(f"https://blockchain.info/unspent?active={address}")

utxo\_set = json.loads(resp.text)["unspent\_outputs"]

# Calculate total balance

total\_balance = sum(utxo["value"] for utxo in utxo\_set)

print(f"Total Balance: {total\_balance} Satoshis")

The script fetches unspent transaction outputs (UTXOs) and sums up their values.

Run this in your environment to get the total balance.

**1.2 Account/Balance Model**

**Question: what is the balance of 0xEeC84548aAd50A465963bB501e39160c58366692. Get the total balance of spendable Weis?**

**python**

from web3 import Web3

# Connect to Ethereum node

rpc\_node = "https://mainnet.infura.io/v3/340ab19d3ab14fcea1d94fb2adde170b"

w3 = Web3(Web3.HTTPProvider(rpc\_node))

# Target account

account = "0xEeC84548aAd50A465963bB501e39160c58366692"

# Fetch and print balance

balance = w3.eth.getBalance(w3.toChecksumAddress(account))

print(f"Balance: {balance} Weis")

**PART II – Unpacking Blockchain Technology**

2.1 SHA256 Exercise

**2.1 SHA256 Exercise**

**Question**: Calculate the SHA256 hash for:

1. "COMP1830-Blockchain for Fintech".
2. "COMP1830-Blockchain for Fintech!" (note the exclamation mark).

python

from hashlib import sha256

data\_1 = "COMP1830-Blockchain for Fintech"

data\_2 = "COMP1830-Blockchain for Fintech!"

# Hash calculations

hash\_1 = sha256(data\_1.encode()).hexdigest()

hash\_2 = sha256(data\_2.encode()).hexdigest()

print(f"Hash 1: {hash\_1}")

print(f"Hash 2: {hash\_2}")

**2.2 Exercise: Use the script provided in “dissecting-blockchain/block.py” to create a blockchain of 4 blocks (1 being the genesis block) that contain the following data respectively. 1. “Blockchain” 2. “Is” 3. “Awesome” What is the hash of the final block?**

1. **"Blockchain".**
2. **"Is".**
3. **"Awesome".**

**patthon**

from hashlib import sha256

class Block:

def \_\_init\_\_(self, index, data, previous\_hash):

self.index = index

self.data = data

self.previous\_hash = previous\_hash

self.hash = self.calculate\_hash()

def calculate\_hash(self):

return

sha256(f"{self.index}{self.data}{self.previous\_hash}".encode()).hexdigest()

# Create blocks

genesis\_block = Block(0, "Blockchain", "0")

block\_2 = Block(1, "Is", genesis\_block.hash)

block\_3 = Block(2, "Awesome", block\_2.hash)

print(f"Genesis Block Hash: {genesis\_block.hash}")

print(f"Block 2 Hash: {block\_2.hash}")

print(f"Block 3 Hash: {block\_3.hash}")

**2.3 Exercise: Try to change the data of the 3rd block in the blockchain to “Is very” and check the validity of blockchain. What should have been the hash of the final block so that the blockchain is valid?**

**Block Tampering**

Task: Update the data of block 3 to "Is very" and verify validity. What must the last block hash be for a valid blockchain?

**Instructions:**

* Update the data of block 3, calculate its hash.
* You should notice that the other hashes (blocks) won't match.
* For the blockchain to be valid, all hashes should be recalculated afterward

**2.4 Peer-to-Peer Network**

**Task**: Run 3 peers, broadcast a tampered blockchain. Are they accepting the data?

Without consensus, tampering is trivial. To prevent this, Proof-of-Work guarantees validation.

**2.5 Proof-of-Work**

Question: Implement PoW and compare with the previous attack.

Use the PoW algorithm to generate a valid blockchain and test resistance to tampering.

Observe that it takes much longer to change blockchain states because of computational hardness.