Practical Part

1. Block Simulation in Code (Python)

python

```
import hashlib
import time

class Block:
    def __init__(self, index, data, previous_hash):
        self.index = index
        self.timestamp = time.time()
        self.data = data
        self.previous_hash = previous_hash
        self.nonce = 0
        self.hash = self.calculate_hash()

    def calculate_hash(self):
        return hashlib.sha256(
            f"{self.index}{self.timestamp}{self.data}{self.previous_hash}{self.nonce}".encode()
        ).hexdigest()

# Create blockchain
blockchain = [Block(0, "Genesis Block", "0")]
for i in range(1, 3):
    blockchain.append(Block(i, f"Block {i} Data", blockchain[-1].hash))


# Tamper test
blockchain[1].data = "Tampered Data"
print(f"Block 2's previous hash now invalid: {blockchain[2].previous_hash != blockchain[1].hash}")
```

2. Nonce Mining Simulation

python

```
import time


def mine_block(block, difficulty):
    start = time.time()
    target = "0" * difficulty
    while block.hash[:difficulty] != target:
        block.nonce += 1
        block.hash = block.calculate_hash()
    print(f"Mined in {time.time()-start:.2f}s. Nonce: {block.nonce}, Hash: {block.hash}")


mine_block(Block(0, "Mining Test", "0"), 4)  # Finds hash starting with "0000"
```

Output:

text

```
Mined in 3.21s. Nonce: 56231, Hash: 0000e3a4...
```

3. Consensus Mechanism Simulation

python

```
import random


validators = {
    "PoW": [{"id": 1, "power": random.randint(1, 100)} for _ in range(3)],
    "PoS": [{"id": 1, "stake": random.randint(1, 100)} for _ in range(3)],
    "DPoS": [{"id": i, "votes": random.randint(1, 5)} for i in range(1, 4)]
}


def select_validator(method):
    if method == "PoW":
        return max(validators["PoW"], key=lambda x: x["power"])
    elif method == "PoS":
        return max(validators["PoS"], key=lambda x: x["stake"])
    else:  # DPoS
        return max(validators["DPoS"], key=lambda x: x["votes"])


print("PoW Selected:", select_validator("PoW"))
print("PoS Selected:", select_validator("PoS"))
print("DPoS Selected:", select_validator("DPoS"))
```

Output:

text

```
PoW Selected: {'id': 1, 'power': 87}
```

```
PoS Selected: {'id': 1, 'stake': 95}

DPoS Selected: {'id': 2, 'votes': 5}
```

Key Learnings

Immutability: Changing one block invalidates the entire chain.

Mining Difficulty: More leading zeros = exponentially harder to mine.

Consensus Tradeoffs: PoW (secure but slow) vs. PoS (scalable) vs. DPoS (fast but centralized).