ASSIGNMENT-006

Smart Contract Development and Unit Testing using Hardhat
Objective
To implement a simple Ethereum smart contract and perform unit testing using Hardhat and Mocha/Chai, ensuring correct functionality through automated tests.
Problem Statement
Develop a basic Personal Portfolio Smart Contract that stores and retrieves portfolio details (name, role, and description).
Write unit tests to validate the contract functions using Hardhat.
Requirements
• Add inline comments explaining the logic
• Run tests and record outputs
• Take screenshot of VS Code while running tests
Software & Tools Required
• VS Code

CODE:

```
import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
import hashlib
import json
import time
from datetime import datetime
from threading import Thread
import random

# ============================================================================
# BLOCKCHAIN CORE CLASSES
# ============================================================================
```

```
class Block:
    """Represents a single block in the blockchain"""

    def __init__(self, index, transactions, previous_hash, timestamp=None, nonce=0):
        self.index = index
        self.transactions = transactions
        self.previous_hash = previous_hash
        self.timestamp = timestamp or datetime.now().isoformat()
        self.nonce = nonce
        self.hash = self.calculate_hash()

    def calculate_hash(self):
        """Calculate SHA-256 hash of the block"""
        block_string = json.dumps({
            "index": self.index,
            "transactions": self.transactions,
            "previous_hash": self.previous_hash,
            "timestamp": self.timestamp,
            "nonce": self.nonce
        }, sort_keys=True)
        return hashlib.sha256(block_string.encode()).hexdigest()

    def mine_block(self, difficulty):
        """Proof of Work - Mine the block"""
        target = "0" * difficulty
        while not self.hash.startswith(target):
            self.nonce += 1
```

```
            self.hash = self.calculate_hash()
            yield self.nonce

    def to_dict(self):
        """Convert block to dictionary"""
        return {
            "index": self.index,
            "timestamp": self.timestamp,
            "transactions": self.transactions,
            "nonce": self.nonce,
            "hash": self.hash,
            "previous_hash": self.previous_hash
        }
```

```
class Blockchain:
    """Represents the entire blockchain"""

    def __init__(self, difficulty=2):
        self.chain = []
        self.pending_transactions = []
        self.difficulty = difficulty
        self.mining_reward = 10

        # Create genesis block
        self.create_genesis_block()

    def create_genesis_block(self):
        """Create the first block in the chain"""
        genesis_block = Block(0, [], "0")
        self.chain.append(genesis_block)

    def get_latest_block(self):
        """Get the last block in the chain"""
        return self.chain[-1]

    def add_transaction(self, transaction):
        """Add a transaction to pending transactions"""
        self.pending_transactions.append(transaction)
        return len(self.chain)

    def mine_pending_transactions(self, miner_address):
        """Mine pending transactions into a new block"""
        # Add mining reward
        self.pending_transactions.append({
            "from": "SYSTEM",
            "to": miner_address,
            "amount": self.mining_reward,
            "timestamp": datetime.now().isoformat()
        })

        block = Block(
            index=len(self.chain),
            transactions=self.pending_transactions,
            previous_hash=self.get_latest_block().hash
        )

        # Mine the block
        for nonce in block.mine_block(self.difficulty):
            yield {
                "status": "mining",
```

```
                "nonce": nonce,
                "target_difficulty": "0" * self.difficulty,
                "current_hash": block.hash[:16] + "..."
            }

        self.chain.append(block)
        self.pending_transactions = []

        yield {
            "status": "complete",
            "block": block.to_dict(),
            "chain_length": len(self.chain)
        }

    def validate_chain(self):
        """Validate the entire blockchain"""
        for i in range(1, len(self.chain)):
            current_block = self.chain[i]
            previous_block = self.chain[i - 1]

            # Check current block's hash
            if current_block.hash != current_block.calculate_hash():
                return False, f"Block {i}: Invalid hash"

            # Check if previous hash matches
            if current_block.previous_hash != previous_block.hash:
                return False, f"Block {i}: Previous hash doesn't match"

            # Check proof of work
            if not current_block.hash.startswith("0" * self.difficulty):
                return False, f"Block {i}: Invalid proof of work"

        return True, "Blockchain is valid!"

    def get_balance(self, address):
        """Calculate balance for an address"""
        balance = 0
        for block in self.chain:
            for transaction in block.transactions:
                if transaction.get("from") == address:
                    balance -= transaction.get("amount", 0)
                if transaction.get("to") == address:
                    balance += transaction.get("amount", 0)
        return balance
```

```
# ============================================================================
# TKINTER GUI APPLICATION
# ============================================================================
```

```
class BlockchainSimulator:
    """Tkinter GUI for blockchain simulation"""

    def __init__(self, root):
        self.root = root
        self.root.title("Blockchain Technology Simulator")
        self.root.geometry("1400x900")
        self.root.configure(bg="#1e1e2e")

        self.blockchain = Blockchain(difficulty=2)
        self.mining_thread = None
```

```
        self.is_mining = False

        self.setup_styles()
        self.create_widgets()
        self.update_blockchain_display()

    def setup_styles(self):
        """Setup ttk styles"""
        style = ttk.Style()
        style.theme_use("clam")

        # Define colors
        bg_color = "#1e1e2e"
        fg_color = "#e0e0e0"
        accent_color = "#00d4ff"

        style.configure("TFrame", background=bg_color)
        style.configure("TLabel", background=bg_color, foreground=fg_color)
        style.configure("TButton", background=accent_color, foreground="black")
        style.map("TButton", background=[("active", "#00a8d4")])
        style.configure("TNotebook", background=bg_color)
        style.configure("TNotebook.Tab", background=bg_color, foreground=fg_color)

    def create_widgets(self):
        """Create main GUI widgets"""
        # Main container
        main_frame = ttk.Frame(self.root)
        main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Create notebook (tabs)
        notebook = ttk.Notebook(main_frame)
        notebook.pack(fill=tk.BOTH, expand=True)

        # Tab 1: Blockchain Visualization
        self.blockchain_frame = ttk.Frame(notebook)
        notebook.add(self.blockchain_frame, text="🔗 Blockchain")
        self.create_blockchain_tab()

        # Tab 2: Mining
        mining_frame = ttk.Frame(notebook)
        notebook.add(mining_frame, text="⛏ Mining")
        self.create_mining_tab(mining_frame)

        # Tab 3: Transactions
        transaction_frame = ttk.Frame(notebook)
        notebook.add(transaction_frame, text="💳 Transactions")
        self.create_transaction_tab(transaction_frame)

        # Tab 4: Validation
        validation_frame = ttk.Frame(notebook)
        notebook.add(validation_frame, text="✓ Validation")
        self.create_validation_tab(validation_frame)

        # Status bar
        self.status_var = tk.StringVar(value="Ready")
        status_bar = ttk.Label(self.root, textvariable=self.status_var, relief=tk.SUNKEN)
        status_bar.pack(fill=tk.X, padx=5, pady=5)

    def create_blockchain_tab(self):
        """Create blockchain visualization tab"""
        # Controls
```

```
        control_frame = ttk.Frame(self.blockchain_frame)
        control_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Button(control_frame, text="Refresh View",
command=self.update_blockchain_display).pack(side=tk.LEFT, padx=5)
        ttk.Button(control_frame, text="Clear Blockchain",
command=self.clear_blockchain).pack(side=tk.LEFT, padx=5)

        # Canvas for visualization
        canvas_frame = ttk.Frame(self.blockchain_frame)
        canvas_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        self.blockchain_canvas = tk.Canvas(
            canvas_frame, bg="#2a2a3a", highlightthickness=0, height=400
        )
        scrollbar = ttk.Scrollbar(canvas_frame, orient=tk.HORIZONTAL,
command=self.blockchain_canvas.xview)
        self.blockchain_canvas.configure(xscrollcommand=scrollbar.set)

        self.blockchain_canvas.pack(fill=tk.BOTH, expand=True)
        scrollbar.pack(fill=tk.X)

        # Stats panel
        stats_frame = ttk.LabelFrame(self.blockchain_frame, text="📊 Blockchain Stats")
        stats_frame.pack(fill=tk.X, padx=5, pady=5)

        self.stats_text = tk.Text(stats_frame, height=6, width=100, bg="#2a2a3a", fg="#00d4ff")
        self.stats_text.pack(fill=tk.X, padx=5, pady=5)
        self.stats_text.config(state=tk.DISABLED)

    def create_mining_tab(self, parent):
        """Create mining tab"""
        control_frame = ttk.LabelFrame(parent, text="⛏ Mining Controls")
        control_frame.pack(fill=tk.X, padx=5, pady=5)

        # Miner address
        ttk.Label(control_frame, text="Miner Address:").grid(row=0, column=0, padx=5, pady=5)
        self.miner_address_var = tk.StringVar(value="Miner_001")
        ttk.Entry(control_frame, textvariable=self.miner_address_var, width=30).grid(row=0,
column=1, padx=5, pady=5)

        # Difficulty
        ttk.Label(control_frame, text="Difficulty:").grid(row=1, column=0, padx=5, pady=5)
        self.difficulty_var = tk.IntVar(value=2)
        difficulty_spinbox = ttk.Spinbox(control_frame, from_=1, to=5,
textvariable=self.difficulty_var, width=10)
        difficulty_spinbox.grid(row=1, column=1, padx=5, pady=5, sticky=tk.W)

        # Buttons
        button_frame = ttk.Frame(control_frame)
        button_frame.grid(row=2, column=0, columnspan=2, padx=5, pady=10)

        self.mine_button = ttk.Button(button_frame, text="Start Mining",
command=self.start_mining)
        self.mine_button.pack(side=tk.LEFT, padx=5)

        self.stop_button = ttk.Button(button_frame, text="Stop Mining",
command=self.stop_mining, state=tk.DISABLED)
        self.stop_button.pack(side=tk.LEFT, padx=5)

        # Mining progress
```

```
        progress_frame = ttk.LabelFrame(parent, text="Mining Progress")
        progress_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        self.mining_text = scrolledtext.ScrolledText(progress_frame, height=20, bg="#2a2a3a",
fg="#00d4ff")
        self.mining_text.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

    def create_transaction_tab(self, parent):
        """Create transactions tab"""
        # Create transaction
        create_frame = ttk.LabelFrame(parent, text="➕ Create New Transaction")
        create_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(create_frame, text="From:").grid(row=0, column=0, padx=5, pady=5)
        self.from_var = tk.StringVar(value="Alice")
        ttk.Entry(create_frame, textvariable=self.from_var, width=30).grid(row=0, column=1,
padx=5, pady=5)

        ttk.Label(create_frame, text="To:").grid(row=1, column=0, padx=5, pady=5)
        self.to_var = tk.StringVar(value="Bob")
        ttk.Entry(create_frame, textvariable=self.to_var, width=30).grid(row=1, column=1,
padx=5, pady=5)

        ttk.Label(create_frame, text="Amount:").grid(row=2, column=0, padx=5, pady=5)
        self.amount_var = tk.StringVar(value="10")
        ttk.Entry(create_frame, textvariable=self.amount_var, width=30).grid(row=2, column=1,
padx=5, pady=5)

        ttk.Button(create_frame, text="Add Transaction",
command=self.add_transaction).grid(row=3, column=0, columnspan=2, pady=10)

        # Pending transactions
        pending_frame = ttk.LabelFrame(parent, text="⏳ Pending Transactions")
        pending_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        self.pending_text = scrolledtext.ScrolledText(pending_frame, height=15, bg="#2a2a3a",
fg="#00d4ff")
        self.pending_text.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)
        self.pending_text.config(state=tk.DISABLED)

    def create_validation_tab(self, parent):
        """Create validation tab"""
        button_frame = ttk.Frame(parent)
        button_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Button(button_frame, text="Validate Blockchain",
command=self.validate_blockchain).pack(side=tk.LEFT, padx=5)
        ttk.Button(button_frame, text="Check Address Balance",
command=self.check_balance).pack(side=tk.LEFT, padx=5)
        ttk.Button(button_frame, text="Tamper Block (Test)",
command=self.tamper_block).pack(side=tk.LEFT, padx=5)

        # Output
        output_frame = ttk.LabelFrame(parent, text="📋 Validation Results")
        output_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        self.validation_text = scrolledtext.ScrolledText(output_frame, height=20, bg="#2a2a3a",
fg="#00d4ff")
        self.validation_text.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

    # ============================================================================
```

```
    # CALLBACK METHODS
    # ========================================================================

    def add_transaction(self):
        """Add a new transaction"""
        try:
            amount = float(self.amount_var.get())
            if amount <= 0:
                messagebox.showerror("Error", "Amount must be positive")
                return

            transaction = {
                "from": self.from_var.get(),
                "to": self.to_var.get(),
                "amount": amount,
                "timestamp": datetime.now().isoformat()
            }

            self.blockchain.add_transaction(transaction)
            self.status_var.set(f"Transaction added: {self.from_var.get()} →
{self.to_var.get()}: {amount}")
            self.update_pending_display()
        except ValueError:
            messagebox.showerror("Error", "Invalid amount")

    def update_pending_display(self):
        """Update pending transactions display"""
        self.pending_text.config(state=tk.NORMAL)
        self.pending_text.delete(1.0, tk.END)

        if not self.blockchain.pending_transactions:
            self.pending_text.insert(tk.END, "No pending transactions\n")
        else:
            for i, tx in enumerate(self.blockchain.pending_transactions, 1):
                self.pending_text.insert(tk.END, f"{i}. {tx['from']} → {tx['to']}:
{tx['amount']} coins\n")
                self.pending_text.insert(tk.END, f"   Time: {tx['timestamp']}\n\n")

        self.pending_text.config(state=tk.DISABLED)

    def start_mining(self):
        """Start mining in a separate thread"""
        if self.is_mining:
            messagebox.showwarning("Warning", "Already mining")
            return

        self.is_mining = True
        self.mine_button.config(state=tk.DISABLED)
        self.stop_button.config(state=tk.NORMAL)

        # Update difficulty
        self.blockchain.difficulty = self.difficulty_var.get()

        mining_thread = Thread(target=self.mining_worker, daemon=True)
        mining_thread.start()

    def mining_worker(self):
        """Worker thread for mining"""
        try:
            miner = self.miner_address_var.get()
            self.mining_text.config(state=tk.NORMAL)
```

```
            self.mining_text.delete(1.0, tk.END)
            self.mining_text.insert(tk.END, f"⛏ Starting to mine block...\n\n")

            start_time = time.time()
            iteration = 0

            for result in self.blockchain.mine_pending_transactions(miner):
                if not self.is_mining:
                    break

                if result["status"] == "mining":
                    iteration += 1
                    if iteration % 1000 == 0:  # Update every 1000 iterations
                        self.mining_text.insert(tk.END, f"Nonce: {result['nonce']}\n")
                        self.mining_text.insert(tk.END, f"Hash: {result['current_hash']}\n\n")
                        self.mining_text.see(tk.END)
                        self.root.update()

                elif result["status"] == "complete":
                    elapsed = time.time() - start_time
                    block = result["block"]

                    self.mining_text.insert(tk.END, f"\n✓ Block Mined Successfully!\n")
                    self.mining_text.insert(tk.END, f"Time: {elapsed:.2f} seconds\n")
                    self.mining_text.insert(tk.END, f"Nonce: {block['nonce']}\n")
                    self.mining_text.insert(tk.END, f"Hash: {block['hash']}\n")
                    self.mining_text.insert(tk.END, f"Block #{block['index']} | Transactions:
{len(block['transactions'])}\n\n")
                    self.mining_text.see(tk.END)

                    self.status_var.set(f"✓ Block #{block['index']} mined in {elapsed:.2f}s by
{miner}")
                    self.update_blockchain_display()
                    self.update_pending_display()

            self.mining_text.config(state=tk.DISABLED)

        except Exception as e:
            self.mining_text.insert(tk.END, f"Error: {str(e)}\n")
            self.mining_text.config(state=tk.DISABLED)

        finally:
            self.is_mining = False
            self.mine_button.config(state=tk.NORMAL)
            self.stop_button.config(state=tk.DISABLED)

    def stop_mining(self):
        """Stop mining"""
        self.is_mining = False
        self.status_var.set("Mining stopped by user")

    def update_blockchain_display(self):
        """Update blockchain visualization"""
        self.blockchain_canvas.delete("all")

        x_offset = 20
        for i, block in enumerate(self.blockchain.chain):
            self.draw_block(x_offset + i * 250, 100, block)

        self.blockchain_canvas.config(scrollregion=self.blockchain_canvas.bbox("all"))
        self.update_stats()
```

```
    def draw_block(self, x, y, block):
        """Draw a single block on canvas"""
        width, height = 220, 180

        # Block background
        color = "#00d4ff" if block.index == 0 else "#4a4a6a"
        self.blockchain_canvas.create_rectangle(x, y, x + width, y + height, fill=color,
outline="#00d4ff", width=2)

        # Block content
        self.blockchain_canvas.create_text(
            x + width/2, y + 15, text=f"Block #{block.index}",
            font=("Arial", 12, "bold"), fill="white"
        )

        self.blockchain_canvas.create_text(
            x + 10, y + 35, text=f"Hash: {block.hash[:12]}...",
            font=("Arial", 9), fill="white", anchor="nw"
        )

        self.blockchain_canvas.create_text(
            x + 10, y + 55, text=f"Prev: {block.previous_hash[:12]}...",
            font=("Arial", 9), fill="white", anchor="nw"
        )

        self.blockchain_canvas.create_text(
            x + 10, y + 75, text=f"Nonce: {block.nonce}",
            font=("Arial", 9), fill="white", anchor="nw"
        )

        self.blockchain_canvas.create_text(
            x + 10, y + 95, text=f"Txns: {len(block.transactions)}",
            font=("Arial", 9), fill="white", anchor="nw"
        )

        self.blockchain_canvas.create_text(
            x + 10, y + 115, text=f"Time: {block.timestamp[11:19]}",
            font=("Arial", 8), fill="white", anchor="nw"
        )

        # Arrow to next block
        if block.index < len(self.blockchain.chain) - 1:
            self.blockchain_canvas.create_line(
                x + width + 5, y + height/2, x + width + 25, y + height/2,
                fill="#00d4ff", width=2, arrow=tk.LAST
            )

    def update_stats(self):
        """Update blockchain stats"""
        self.stats_text.config(state=tk.NORMAL)
        self.stats_text.delete(1.0, tk.END)

        stats = f"""
BLOCKCHAIN STATISTICS:
═══════════════════════════════════════════════════
Total Blocks: {len(self.blockchain.chain)}
Difficulty Level: {self.blockchain.difficulty}
Mining Reward: {self.blockchain.mining_reward} coins
Pending Transactions: {len(self.blockchain.pending_transactions)}
Latest Block Hash: {self.blockchain.get_latest_block().hash}
```

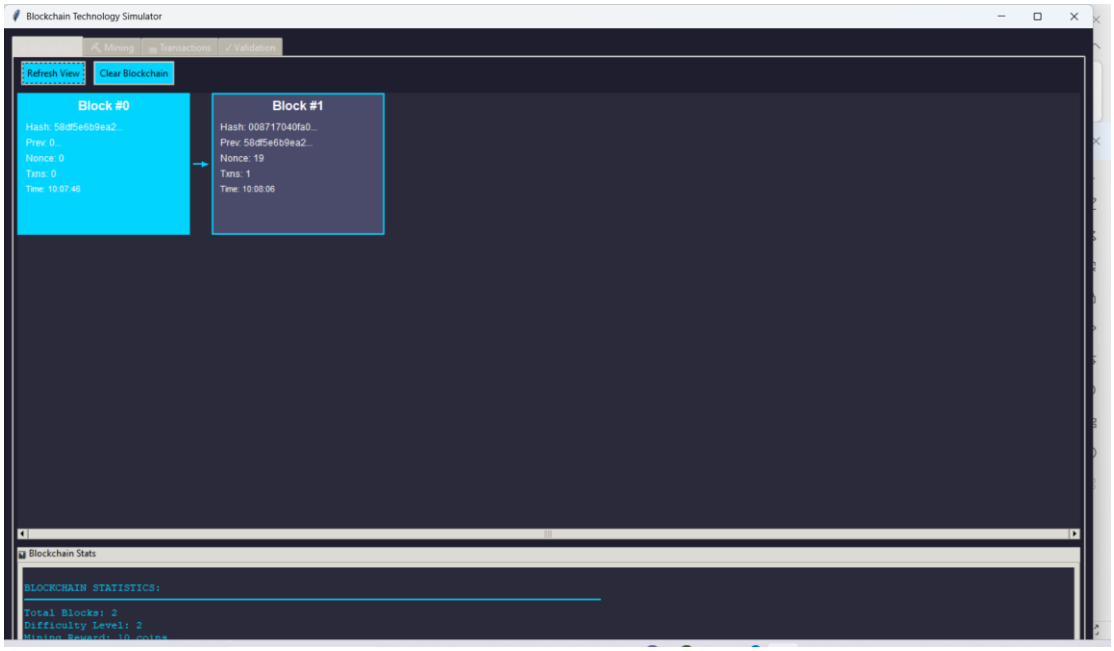
```
Latest Block Nonce: {self.blockchain.get_latest_block().nonce}
        """

        self.stats_text.insert(tk.END, stats)
        self.stats_text.config(state=tk.DISABLED)

    def validate_blockchain(self):
        """Validate the blockchain"""
        validation_frame = self.root.nametowidget(self.root.winfo_children()[0])
        validation_text = None

        for widget in validation_frame.winfo_children():
            if isinstance(widget, ttk.Frame):
                for child in widget.winfo_children():
                    if isinstance(child, scrolledtext.ScrolledText) and "validation" in
str(child):
                        validation_text = child
                        break

        # Find validation text widget
        for widget in self.root.winfo_children():
            if isinstance(widget, ttk.Frame):
                self._find_validation_text(widget)

        is_valid, message = self.blockchain.validate_chain()

        # Update validation text if found
        for notebook in self.root.winfo_children():
            if isinstance(notebook, ttk.Notebook):
                for frame in notebook.winfo_children():
                    for widget in frame.winfo_children():
                        if isinstance(widget, scrolledtext.ScrolledText):
                            if widget not in [self.mining_text, self.pending_text,
self.stats_text]:
                                widget.config(state=tk.NORMAL)
                                widget.delete(1.0, tk.END)

                                result = "✓ VALID" if is_valid else "✗ INVALID"
                                widget.insert(tk.END, f"Blockchain Status: {result}\n\n")
                                widget.insert(tk.END, f"Message: {message}\n\n")
                                widget.insert(tk.END, f"Total Blocks:
{len(self.blockchain.chain)}\n")

                                if is_valid:
                                    widget.insert(tk.END, "\n✓ All blocks have valid hashes\n")
                                    widget.insert(tk.END, "✓ All previous hashes match\n")
                                    widget.insert(tk.END, "✓ All proof of work requirements
met\n")

                                widget.config(state=tk.DISABLED)

        self.status_var.set(f"Validation: {message}")

    def check_balance(self):
        """Check address balance"""
        balance_window = tk.Toplevel(self.root)
        balance_window.title("Check Balance")
        balance_window.geometry("400x200")
        balance_window.configure(bg="#1e1e2e")

        ttk.Label(balance_window, text="Enter Address:").pack(padx=10, pady=10)
```

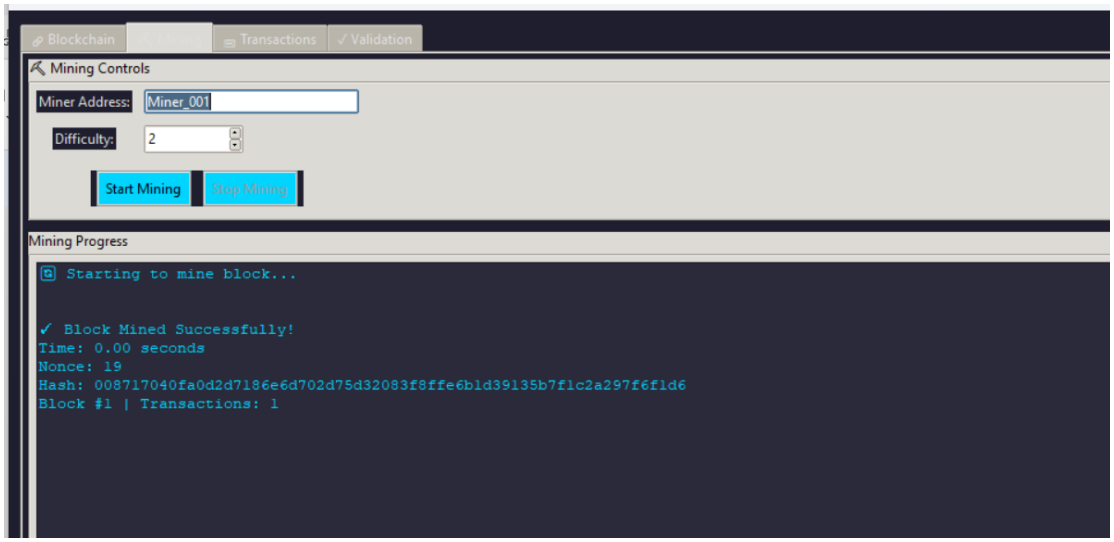
```
        address_var = tk.StringVar()
        ttk.Entry(balance_window, textvariable=address_var, width=40).pack(padx=10, pady=5)

        result_text = scrolledtext.ScrolledText(balance_window, height=8, bg="#2a2a3a",
fg="#00d4ff")
        result_text.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
        result_text.config(state=tk.DISABLED)

        def check():
            address = address_var.get()
            if not address:
                messagebox.showwarning("Warning", "Enter an address")
                return

            balance = self.blockchain.get_balance(address)
            result_text.config(state=tk.NORMAL)
            result_text.delete(1.0, tk.END)
            result_text.insert(tk.END, f"Address: {address}\n")
            result_text.insert(tk.END, f"Balance: {balance} coins\n\n")

            # Show transactions
            result_text.insert(tk.END, "Transactions:\n")
            for block in self.blockchain.chain:
                for tx in block.transactions:
                    if tx.get("from") == address or tx.get("to") == address:
                        result_text.insert(tk.END, f"  {tx['from']} → {tx['to']}:
{tx['amount']}\n")

            result_text.config(state=tk.DISABLED)

        ttk.Button(balance_window, text="Check", command=check).pack(pady=10)

    def tamper_block(self):
        """Tamper with a block for testing"""
        if len(self.blockchain.chain) < 2:
            messagebox.showwarning("Warning", "Need at least 2 blocks")
            return

        # Tamper with second block
        self.blockchain.chain[1].transactions.append({
            "from": "Hacker",
            "to": "Hacker",
            "amount": 1000,
            "timestamp": datetime.now().isoformat()
        })

        messagebox.showinfo("Info", "Block tampered! Validation will now fail.")
        self.status_var.set("Block tampered - validation will fail")

    def clear_blockchain(self):
        """Clear blockchain"""
        if messagebox.askyesno("Confirm", "Clear entire blockchain?"):
            self.blockchain = Blockchain(difficulty=self.blockchain.difficulty)
            self.update_blockchain_display()
            self.update_pending_display()
            self.status_var.set("Blockchain cleared")

    def _find_validation_text(self, widget):
        """Helper to find validation text widget"""
        pass
```

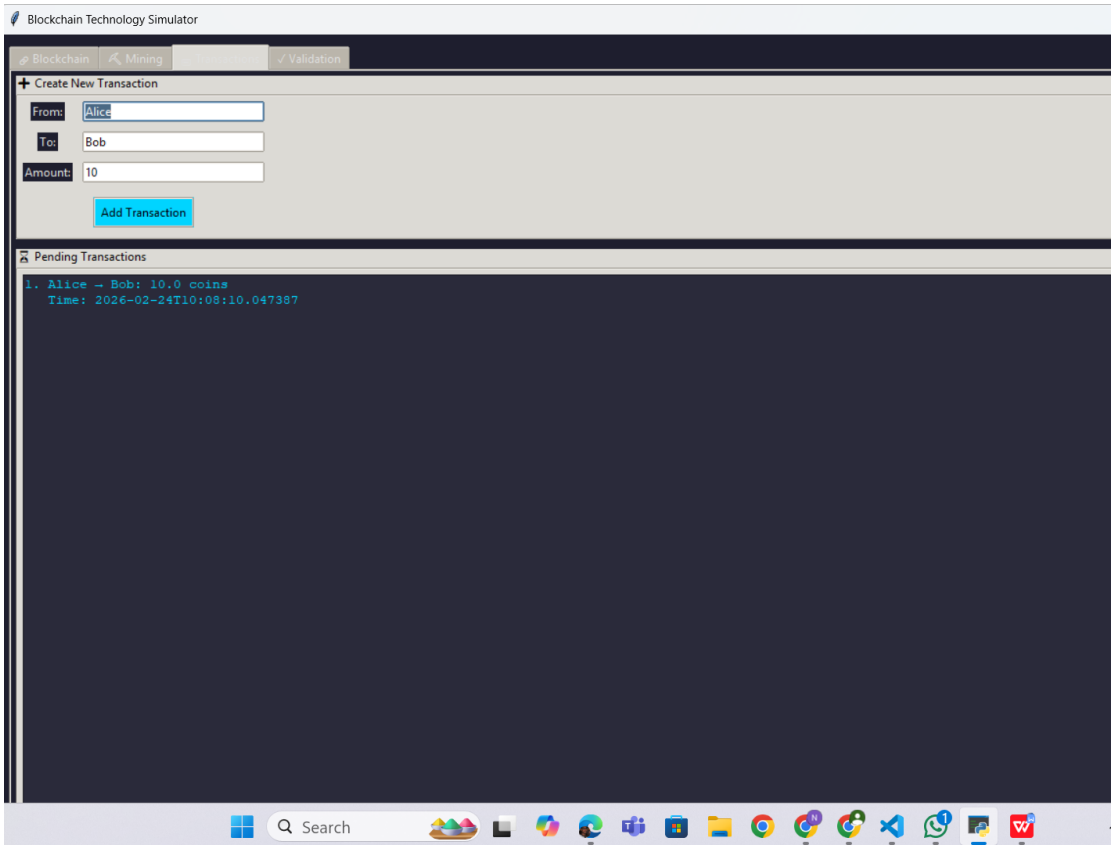
```
# ============================================================================
# MAIN
# ============================================================================
```

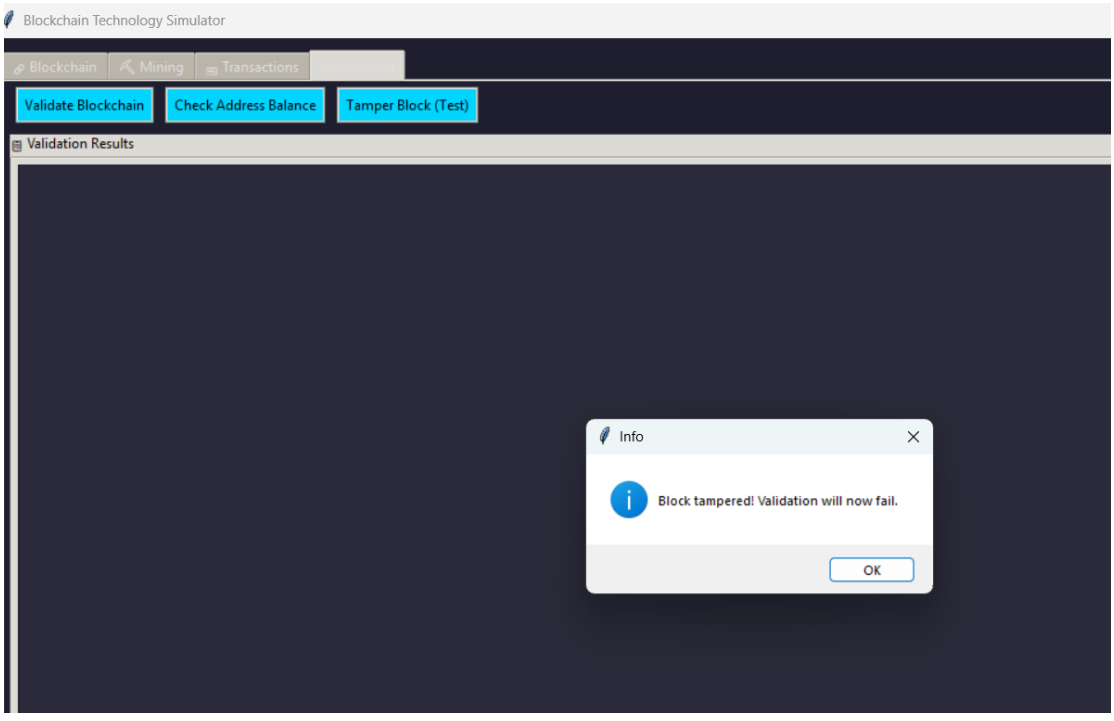
```
if __name__ == "__main__":
    root = tk.Tk()
    app = BlockchainSimulator(root)
    root.mainloop()
```

OUTPUT:






Blockchain Technology Simulator
Blockchain
Mining
Validation
Create New Transaction
From:
Alice
To:
Bob
Amount:
10
Add Transaction
Pending Transactions
1. Alice → Bob: 10.0 coins
Time: 2026-02-24T10:08:10.047387
Search


Blockchain Technology Simulator
Blockchain
Mining
Transactions
Validate Blockchain
Check Address Balance
Tamper Block (Test)
Validation Results
Info
Block tampered! Validation will now fail.
OK