Hash Function

```
def manual_hash(data):
    # Simple manual hash using sum of character codes and mod operation
    hash_value = 0
    for char in data:
        hash_value = (hash_value + ord(char)) % 256  # Limiting hash to 256
    return hex(hash_value)[2:]
```

```
def manual_hash(data):  1 usage
    # Simple manual hash using sum of character codes and mod operation
    hash_value = 0
    for char in data:
        hash_value = (hash_value + ord(char)) % 256  # Limiting hash to 256
    return hex(hash_value)[2:]
```

1. **manual_hash(data):** This defines our custom hashing function that takes a string (data) as input.
2. **hash_value = 0** Initializes the hash value to 0.
3. **for char in data:** Loops through every character in the string.
4. **ord(char)** Converts the character to its ASCII value.
5. **(hash_value + ord(char)) % 256** Adds the ASCII value to the hash and keeps it within a range of 256.
6. **hex(hash_value)[2:]** Converts the hash value into a hexadecimal string (removing the "0x" prefix).

Block Class

```
class Block:
    def __init__(self, data, previous_hash):
        self.timestamp = datetime.datetime.now()
        self.data = data
        self.previous_hash = previous_hash
        self.hash = self.calculate_hash()
```

```
# Block class
class Block:  2 usages
    def __init__(self, data, previous_hash):
        self.timestamp = datetime.datetime.now()
        self.data = data
        self.previous_hash = previous_hash
        self.hash = self.calculate_hash()
```

7. **class Block:** Defines the blueprint for a block in our blockchain.
8. **__init__** Initializes the block with data, a timestamp, and links to the previous block.
9. **self.timestamp = datetime.datetime.now()** Captures the current date and time.
10. **self.data = data** Stores the dummy data in the block.
11. **self.previous_hash = previous_hash** Saves the hash of the previous block.

12. **self.hash = self.calculate_hash()** Calls the calculate_hash method to compute the block's unique hash.

```
def calculate_hash(self):
    block_data = f\"{self.timestamp}{self.data}{self.previous_hash}\"
    return manual_hash(block_data)
```

```
def calculate_hash(self):  2 usages
    block_data = f"{self.timestamp}{self.data}{self.previous_hash}"
    return manual_hash(block_data)
```

13. **calculate_hash()** Combines the block's data into a single string.
14. **manual_hash(block_data)** Uses our custom hash function to calculate the block's unique hash.

Blockchain Class

```
class Blockchain:
    def __init__(self):
        self.chain = [self.create_genesis_block()]
```

```
# Blockchain class
class Blockchain:  1 usage
    def __init__(self):
        self.chain = [self.create_genesis_block()]
```

15. **class Blockchain:** Defines the structure of our blockchain.
16. **self.chain = [self.create_genesis_block()]** Initializes the blockchain with a single "genesis block."

```
def create_genesis_block(self):
    return Block(\"Genesis Block\", \"0\")
```

```
def create_genesis_block(self):  1 usage
    return Block( data: "Genesis Block",  previous_hash: "0")
```

17. **create_genesis_block()** Creates the very first block with the label "Genesis Block" and no previous hash ("0").

```
def add_block(self, data):
    previous_block = self.chain[-1]
```

```
    new_block = Block(data, previous_block.hash)
    self.chain.append(new_block)
```

```
def add_block(self, data):  3 usages
    previous_block = self.chain[-1]
    new_block = Block(data, previous_block.hash)
    self.chain.append(new_block)
```

18. **add_block(data)** Adds a new block to the chain.
19. **previous_block = self.chain[-1]** Fetches the last block in the chain.
20. **Block(data, previous_block.hash)** Creates a new block, linking it to the last block's hash.
21. **self.chain.append(new_block)** Appends the new block to the chain.

```
def is_chain_valid(self):
    for i in range(1, len(self.chain)):
        current_block = self.chain[i]
        previous_block = self.chain[i - 1]
```

```
def is_chain_valid(self):  2 usages (2 dynamic)
    for i in range(1, len(self.chain)):
        current_block = self.chain[i]
        previous_block = self.chain[i - 1]
```

22. **is_chain_valid()** Checks if the blockchain is valid.
23. **range(1, len(self.chain))** Loops through all blocks, skipping the genesis block.

```
        if current_block.hash != current_block.calculate_hash():
            return False
```

```
# Validate block hash
if current_block.hash != current_block.calculate_hash():
    return False
```

24. Validates each block's hash by recalculating it and comparing with the stored value.

```
        if current_block.previous_hash != previous_block.hash:
            return False
```

```
    # Validate previous hash link
    if current_block.previous_hash != previous_block.hash:
        return False
```

25. Checks if the previous_hash matches the hash of the previous block.

Blockchain Explorer GUI

```
class BlockchainExplorer:
    def __init__(self, blockchain):
        self.blockchain = blockchain
        self.root = tk.Tk()
        self.root.title(\"Blockchain Explorer\")
        self.create_gui()
```

```
class BlockchainExplorer:  1 usage
    def __init__(self, blockchain):
        self.blockchain = blockchain
        self.root = tk.Tk()
        self.root.title("Blockchain Explorer")
        self.create_gui()
```

26. **class BlockchainExplorer:** Creates a graphical interface for viewing the blockchain.
27. **self.blockchain = blockchain** Links the blockchain to the explorer.
28. **tk.Tk()** Initializes a new window.
29. **self.root.title(\"Blockchain Explorer\")** Sets the window title.
30. **self.create_gui()** Sets up the interface layout.

```
    def create_gui(self):
        self.tree = ttk.Treeview(self.root, columns=(\"#1\", \"#2\",
\"#3\", \"#4\"), show=\"headings\")
```

```
def create_gui(self):  1 usage
    self.tree = ttk.Treeview(self.root, columns=("#1", "#2", "#3", "#4"), show="headings")
```

31. **ttk.Treeview** Creates a table-like widget for displaying the blocks.
32. **columns=(\"#1\", \"#2\", \"#3\", \"#4\")** Specifies 4 columns: block address, timestamp, data, and validation status.

```
        self.tree.heading(\"#1\", text=\"Block Address\")
        self.tree.heading(\"#2\", text=\"Timestamp\")
        self.tree.heading(\"#3\", text=\"Data\")
        self.tree.heading(\"#4\", text=\"Validation Status\")
```

```
        self.tree.pack(fill=tk.BOTH, expand=True)
```

```
self.tree.heading("#1", text="Block Address")
self.tree.heading("#2", text="Timestamp")
self.tree.heading("#3", text="Data")
self.tree.heading("#4", text="Validation Status")
self.tree.pack(fill=tk.BOTH, expand=True)

self.update_tree_view()
```

33. Sets column headers and arranges the table to expand and fill the window.

```
        self.update_tree_view()
```

```
self.update_tree_view()
```

34. Calls update_tree_view() to populate the table with block data.

```
    def update_tree_view(self):
        for block in self.blockchain.chain:
            validation_status = \"Valid\" if
self.blockchain.is_chain_valid() else \"Invalid\"
```

```
def update_tree_view(self):  1 usage
    for block in self.blockchain.chain:
        validation_status = "Valid" if self.blockchain.is_chain_valid() else "Invalid"
        self.tree.insert( parent: "", index: "end", values=(
            block.hash,
            block.timestamp,
            block.data,
            validation_status,
        ))
```

35. Loops through the blocks and checks their validation status.

```
            self.tree.insert(\"\", \"end\", values=(\n
block.hash,\n                block.timestamp,\n
block.data,\n                validation_status,\n            ))\n```
```

```
self.tree.insert( parent: "", index: "end", values=(
    block.hash,
    block.timestamp,
    block.data,
    validation_status,
))
```

36. Inserts each block's information into the table.

```
```python
    def run(self):
        self.root.mainloop()
```

```
def run(self):  1 usage
    self.root.mainloop()
```

37. **run()** Starts the GUI application.

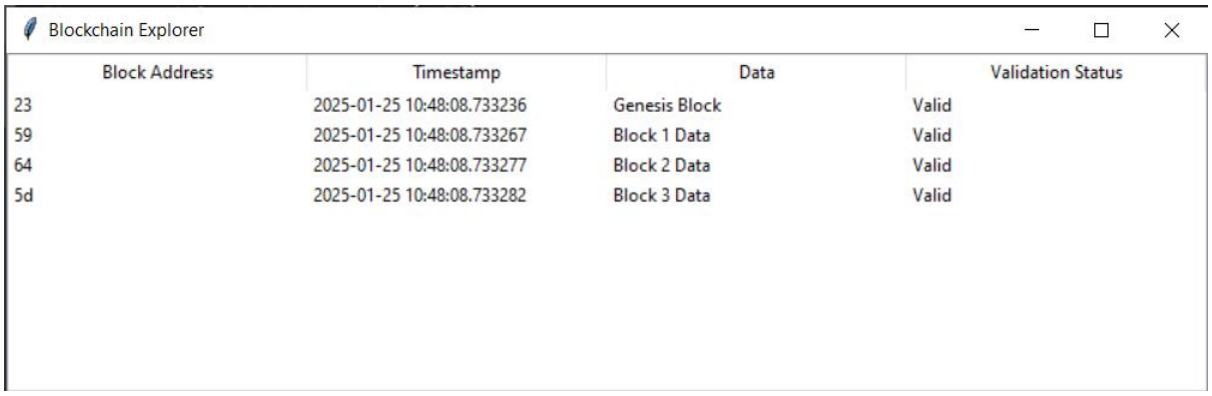
Main Script

```
if __name__ == \"__main__\":\n    blockchain = Blockchain()\n
blockchain.add_block(\"Block 1 Data\")\n    blockchain.add_block(\"Block 2
Data\")\n    blockchain.add_block(\"Block 3 Data\")\n\n    explorer =
BlockchainExplorer(blockchain)\n    explorer.run()
```

```
# Main script
if __name__ == "__main__":
    blockchain = Blockchain()
    blockchain.add_block("Block 1 Data")
    blockchain.add_block("Block 2 Data")
    blockchain.add_block("Block 3 Data")

    explorer = BlockchainExplorer(blockchain)
    explorer.run()
```

38. **blockchain = Blockchain()** Creates a new blockchain.
39. **add_block()** Adds dummy data to the blockchain.
40. **BlockchainExplorer(blockchain)** Initializes the GUI with our blockchain.
41. **explorer.run()** Launches the blockchain explorer.
```


Blockchain Explorer
Block Address
Timestamp
Data
Validation Status
23
2025-01-25 10:48:08.733236
Genesis Block
Valid
59
2025-01-25 10:48:08.733267
Block 1 Data
Valid
64
2025-01-25 10:48:08.733277
Block 2 Data
Valid
5d
2025-01-25 10:48:08.733282
Block 3 Data
Valid