Syntax for Key Concepts

1. Structs (Defining a Block)

```
// Block represents a block in the blockchain
type Block struct
{
  transactions [ ] string
  prevPointer  *Block
  prevHash     string
  currentHash  string
}
```

2. Pointers (Linking Blocks)

```
var chainHead *Block // Points to the latest block
```

3. Maps (Storing Transaction Data)

```
Spender := make(map[string]int)
Receiver := make(map[string]int)
```

4. Slices (Handling Multiple Transactions)

```
transactions := []string{"Tx1", "Tx2"}
```

5. Functions (Example: Calculate Balance)

```
func CalculateBalance ( userName string, chainHead *Block ) int
{
  balance := 0
  tempBlock := chainHead

  for tempBlock != nil
```

```
  {
    balance += tempBlock.Receiver[userName] - tempBlock.Spender[userName]
    tempBlock = tempBlock.prevPointer
  }
  return balance
}
```

6. Loops (Iterating Over Blockchain)

```
 for tempBlock != nil
{
  fmt.Println(tempBlock.transactions)
  tempBlock = tempBlock.prevPointer
}
```

7. Hashing (SHA-256)

```
import (
    "crypto/sha256"
    "encoding/hex"
    "fmt"
)
```

```
// CalculateHash computes the hash of a block: prevHash + data
func CalculateHash ( block *Block ) string {
  data := block.prevHash
  for _ , transaction :=  range block.transactions {
    data += transaction
  }
  hash := sha256.Sum256( [ ] byte ( data ) )
  return hex.EncodeToString( hash [ : ] )     //convert the array of 32 bytes into a string
}
```

8. Conditionals (Blockchain Verification)

```
// Check if the calculated hash matches the stored hash
    if calculatedHash != currentBlock.currentHash {
        fmt.Println("Blockchain is compromised! Hash mismatch in block.")
        return
    }
```

9. Printing & Debugging

```
fmt.Println("Blockchain is valid!")
```

Q3: Complete following incomplete functions (highlighted in bold) in GoLang **[4+5 marks]**

Part 1:

```
type Block struct {
        Spender     map[string]int //Spender is an array of integers in which the indexes are strings
        Receiver    map[string]int
        PrevPointer *Block
        PrevHash    string
        CurrentHash string
}
func CalculateBalance(userName string, chainHead *Block) int {
//calculate balance of a specific user


        var balance = 0
        var amountSpend = 0
        var amountReceived = 0
        var tempBlock = chainHead
        for tempBlock != nil {
                amountReceived += tempBlock.Receiver[userName]
                amountSpend += tempBlock.Spender[userName]
                tempBlock = tempBlock.PrevPointer
        }
        balance = amountReceived - amountSpend
        return balance
```

```
func InsertBlock(transactionsToInsert []string, chainHead *Block) *Block {
//inserting new Block
        if chainHead == nil {
                chainHead = &Block{} //creating new block
                chainHead.transactions = transactionsToInsert
                chainHead.prevPointer=nil
                chainHead.prevHash=""
                chainHead.currentHash = CalculateHash(chainHead)
                return chainHead









        else{

                var newBlock *Block
                newBlock = &Block{}

                newBlock.transactions = transactionsToInsert
                newBlock.prevPointer = chainHead
                newBlock.prevHash = CalculateHash(chainHead)
                newBlock.currentHash = CalculateHash(newBlock)

                chainHead = newBlock

                return chainHead
```

```
// VerifyChain checks if the blockchain is valid
func VerifyChain(chainHead *Block) {
  // Start from the head of the chain
  currentBlock := chainHead
```

```
    // Traverse the blockchain
    for currentBlock != nil {
        // Recalculate the hash of the current block
        calculatedHash := CalculateHash(currentBlock)

        // Check if the calculated hash matches the stored hash
        if calculatedHash != currentBlock.currentHash {
            fmt.Println("Blockchain is compromised! Hash mismatch in block.")
            return
        }

        // Check if the previous block exists
        if currentBlock.prevPointer != nil {
            // Verify that the prevHash of the current block matches the currentHash of the previous block
            if currentBlock.prevHash != currentBlock.prevPointer.currentHash {
                fmt.Println("Blockchain is compromised! Previous hash mismatch.")
                return
            }
        }

        // Move to the previous block
        currentBlock = currentBlock.prevPointer
    }

    // If all blocks are valid
    fmt.Println("Blockchain is unchanged and valid.")
}
```

```
// CalculateHash computes the hash of a block: prevHash + data
func CalculateHash(block *Block) string {
```

```
    data := block.prevHash
    for _, transaction := range block.transactions {
        data += transaction
    }
    hash := sha256.Sum256([]byte(data))
    return hex.EncodeToString(hash[:])      //convert the array of 32 bytes into a string
}
```

```
// ChangeBlock modifies a transaction in a block
func ChangeBlock(oldTrans string, newTrans string, chainHead *Block) {
    currentBlock := chainHead
    for currentBlock != nil {
        for i, transaction := range currentBlock.transactions {
            if transaction == oldTrans {
                currentBlock.transactions[i] = newTrans
                currentBlock.currentHash = CalculateHash(currentBlock)
                fmt.Println("Transaction changed successfully!")
                return
            }
        }
        currentBlock = currentBlock.prevPointer
    }
    fmt.Println("Transaction not found in the blockchain.")
}
```

```
// ListBlocks displays all blocks in the blockchain
func ListBlocks(chainHead *Block) {
    currentBlock := chainHead
    blockNumber := 1
    for currentBlock != nil {
        fmt.Printf("Block %d:\n", blockNumber)
```

```
        fmt.Println("Transactions:", currentBlock.transactions)
        fmt.Println("Previous Hash:", currentBlock.prevHash)
        fmt.Println("Current Hash:", currentBlock.currentHash)
        fmt.Println("----------------------")
        currentBlock = currentBlock.prevPointer
        blockNumber++
    }
}
```