HYPERLEDGER
MEETUP

# Smart Contracts

By: Moolshanker Kothari

# About Me

- Email:
  - moolkothari@hotmail.com
- Twitter:
  - @moolkothari
- LinkedIn:
- http://www.linkedin.com/in/moolshankerkothari
- Facebook:
  - https://www.facebook.com/Blockchian-Development-334736094044864

# Agenda

- What is Smart Contract (Chaincode)
- Writing chaincode using Go
- Compiling and Deploying Chaincode
- Running and Testing the Chaincode
- Developing an application using SDK

# Smart Contract

- In Hyperledger it is called Chaincode
- Smart contracts are the bread and butter of blockchain technology
- Smart contracts are the digitized business logic used to help you exchange any asset of value (money, real estate, retail products, etc)
- A Smart Contract typically handles business logic agreed to by members of the network
- Smart Contracts automatically execute transactions and record information onto the ledger.

# Development Languages Options

- The Hyperledger Fabric chaincode can be programmed in

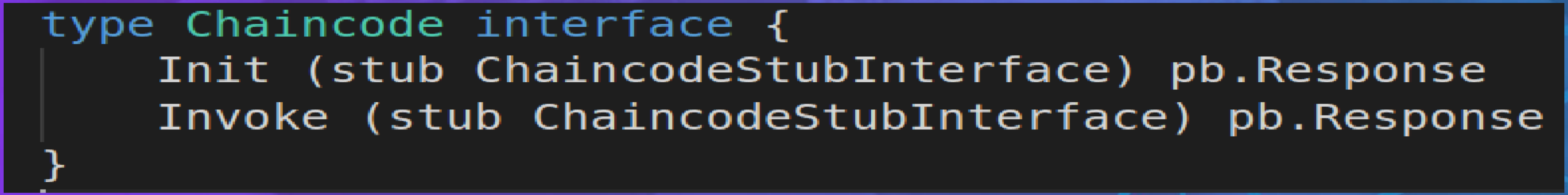
1. Go
2. Node.js
3. Java

https://stackoverflow.com/questions/54603029/hyperledger-fabric-chaincode-development-language-nodejs-java-or-go

# Writing chaincode using Go

- Every chaincode needs to implement a Chaincode interface.
- There are two methods defined in the interface

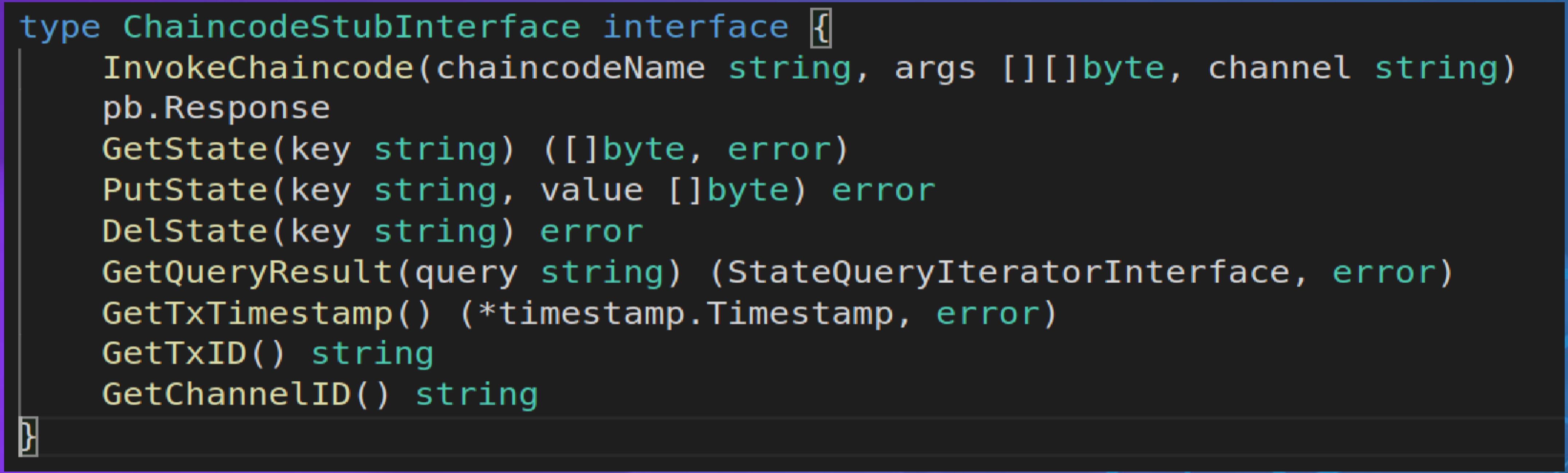
1. **Init**: called when the chaincode is instantiated by the blockchain network
2. **Invoke**: called when the client invokes a specific function to process the transaction proposal



```
type Chaincode interface {
    Init (stub ChaincodeStubInterface) pb.Response
    Invoke (stub ChaincodeStubInterface) pb.Response
}
```

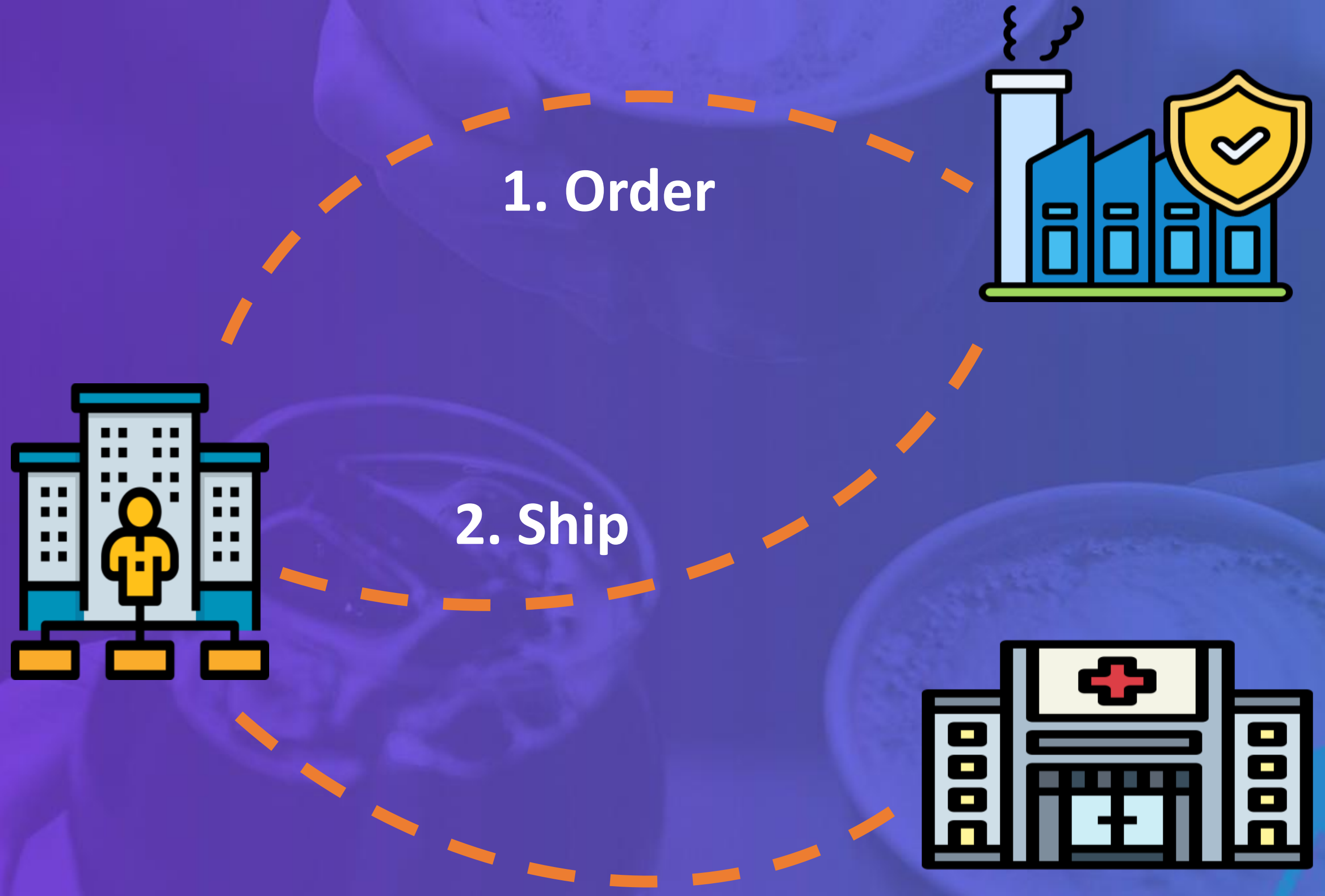
# ChaincodeStubInterface

- ChaincodeStubInterface provides the API for apps to access and modify their ledgers.
- Some important APIs are:

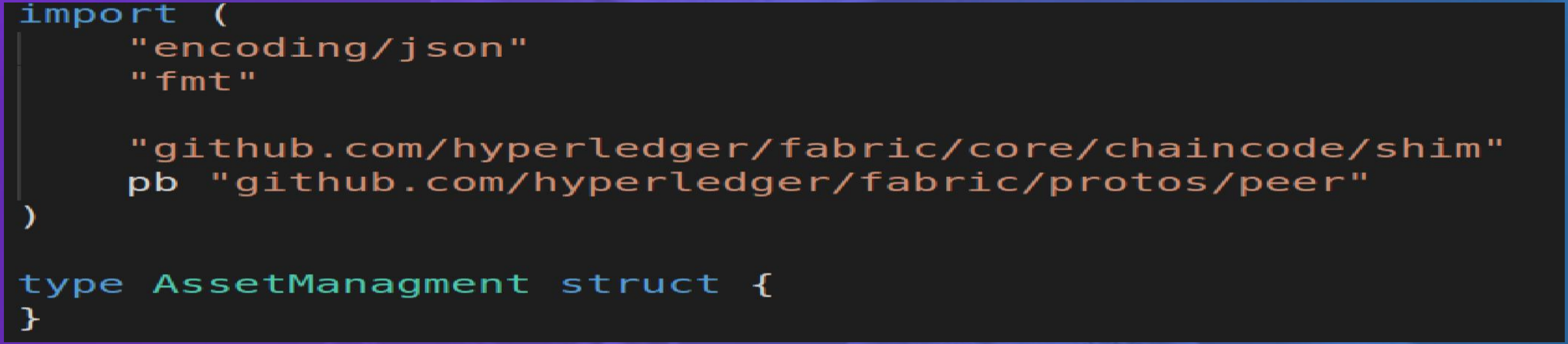


```
type ChaincodeStubInterface interface {
    InvokeChaincode(chaincodeName string, args [][]byte, channel string)
    pb.Response
    GetState(key string) ([]byte, error)
    PutState(key string, value []byte) error
    DelState(key string) error
    GetQueryResult(query string) (StateQueryIteratorInterface, error)
    GetTxTimestamp() (*timestamp.Timestamp, error)
    GetTxID() string
    GetChannelID() string
}
```

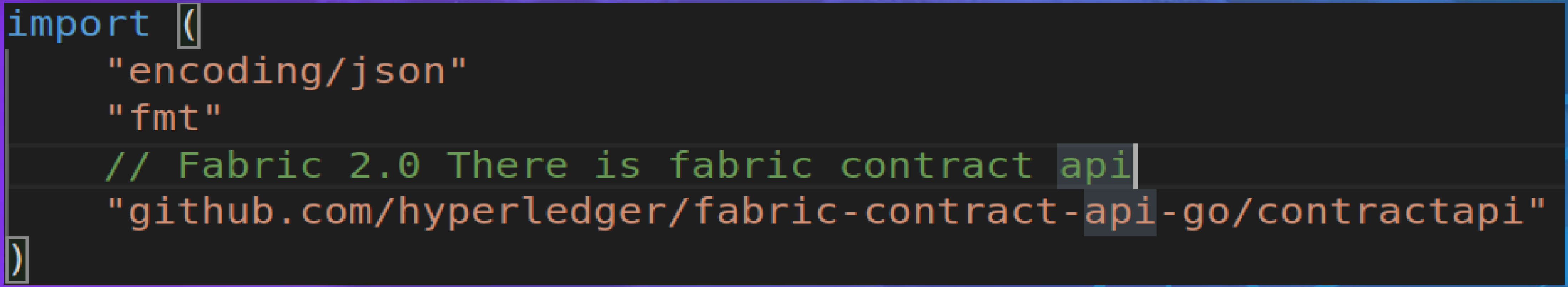
# Hospital Asset Managment



# Step 1

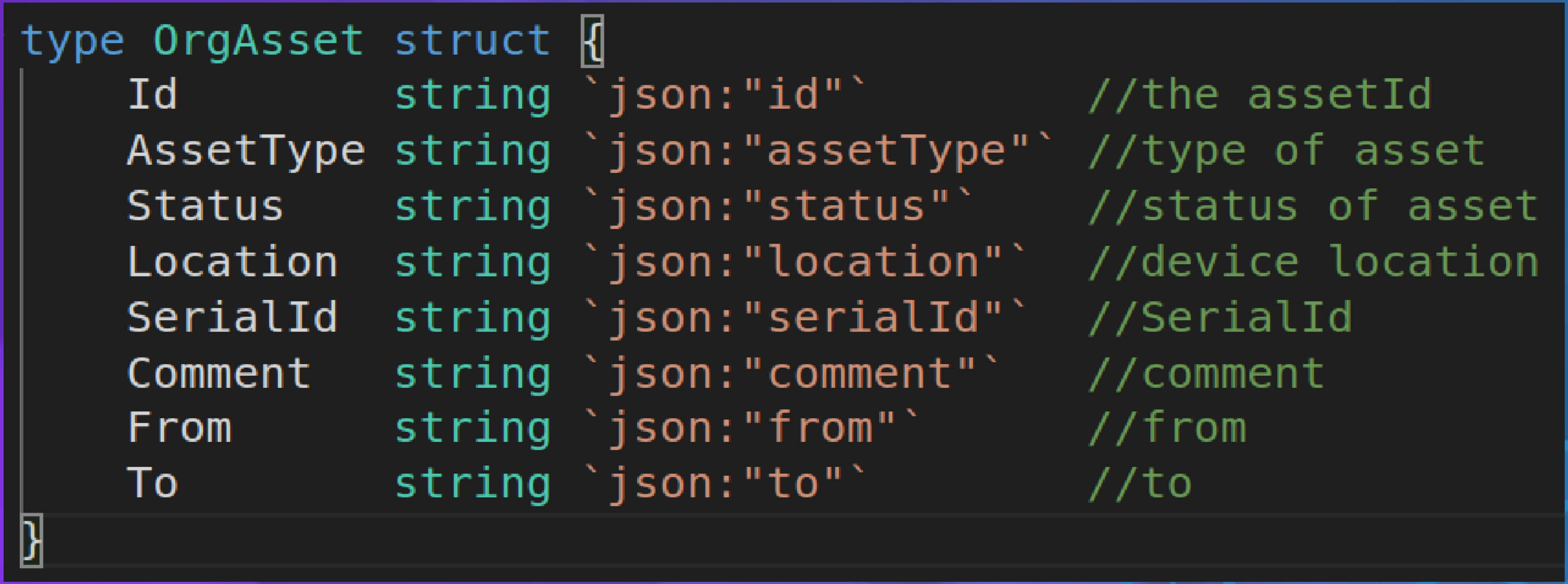


```
import (
	"encoding/json"
	"fmt"

	"github.com/hyperledger/fabric/core/chaincode/shim"
	pb "github.com/hyperledger/fabric/protos/peer"
)

type AssetManagment struct {
}
```



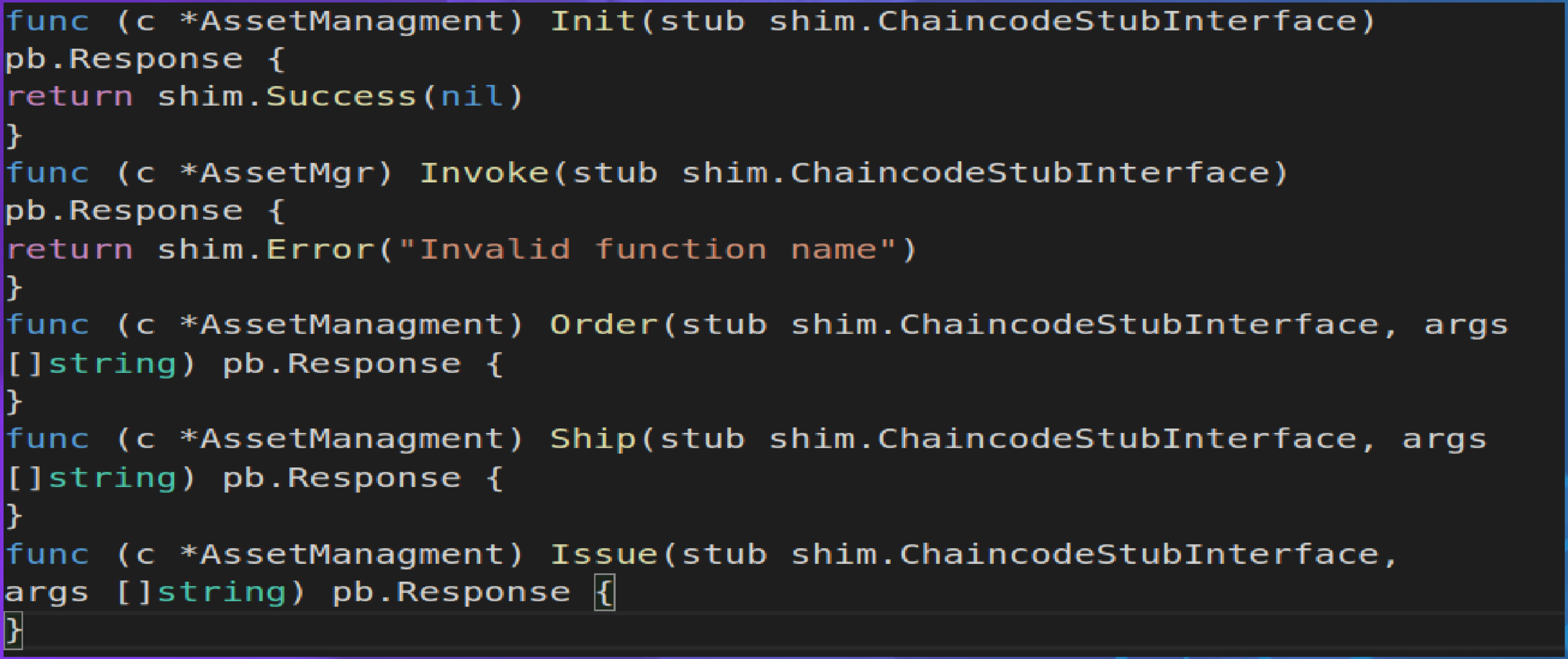
```
import (
	"encoding/json"
	"fmt"
	// Fabric 2.0 There is fabric contract api
	"github.com/hyperledger/fabric-contract-api-go/contractapi"
)
```

# Step 2



```
type OrgAsset struct {
    Id        string `json:"id"`        //the assetId
    AssetType string `json:"assetType"` //type of asset
    Status    string `json:"status"`    //status of asset
    Location  string `json:"location"`  //device location
    SerialId  string `json:"serialId"`  //SerialId
    Comment   string `json:"comment"`   //comment
    From      string `json:"from"`      //from
    To        string `json:"to"`        //to
}
```

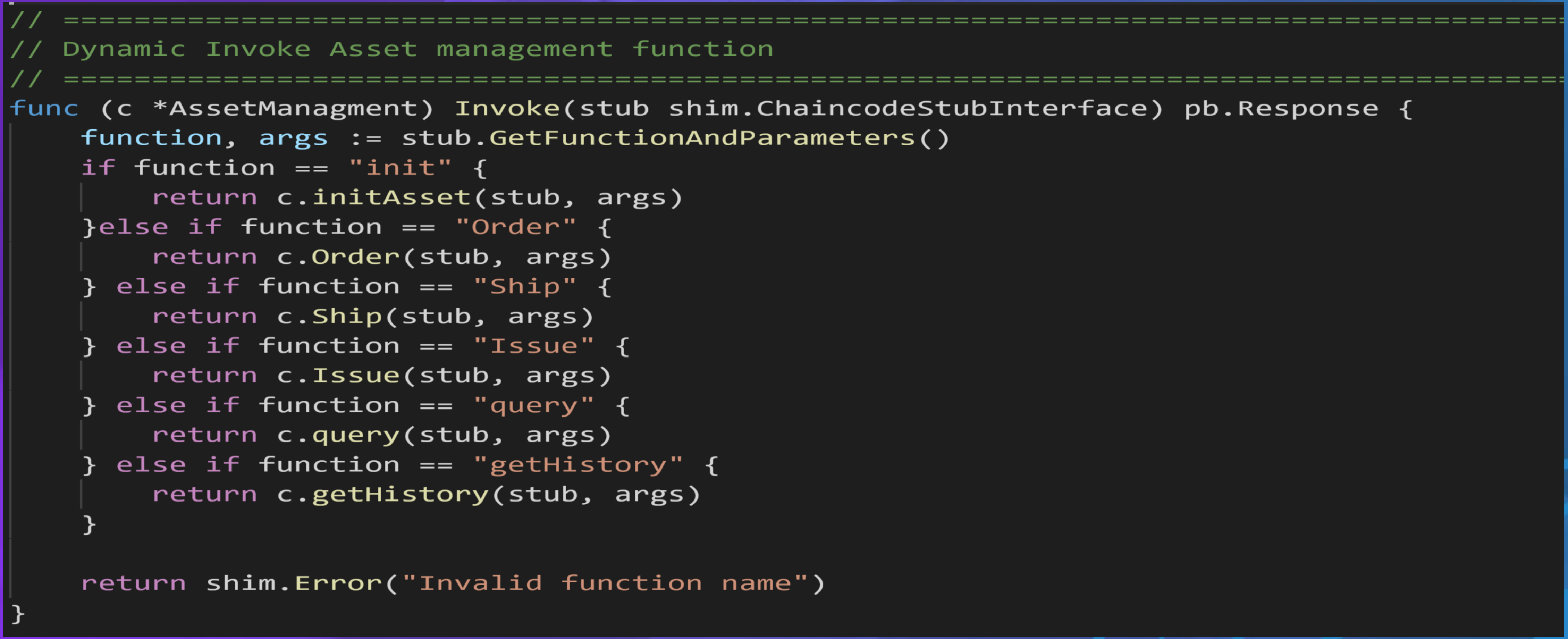
# Step 3



```
func (c *AssetManagment) Init(stub shim.ChaincodeStubInterface)
pb.Response {
return shim.Success(nil)
}
func (c *AssetMgr) Invoke(stub shim.ChaincodeStubInterface)
pb.Response {
return shim.Error("Invalid function name")
}
func (c *AssetManagment) Order(stub shim.ChaincodeStubInterface, args
[]string) pb.Response {
}
func (c *AssetManagment) Ship(stub shim.ChaincodeStubInterface, args
[]string) pb.Response {
}
func (c *AssetManagment) Issue(stub shim.ChaincodeStubInterface,
args []string) pb.Response {
}
```

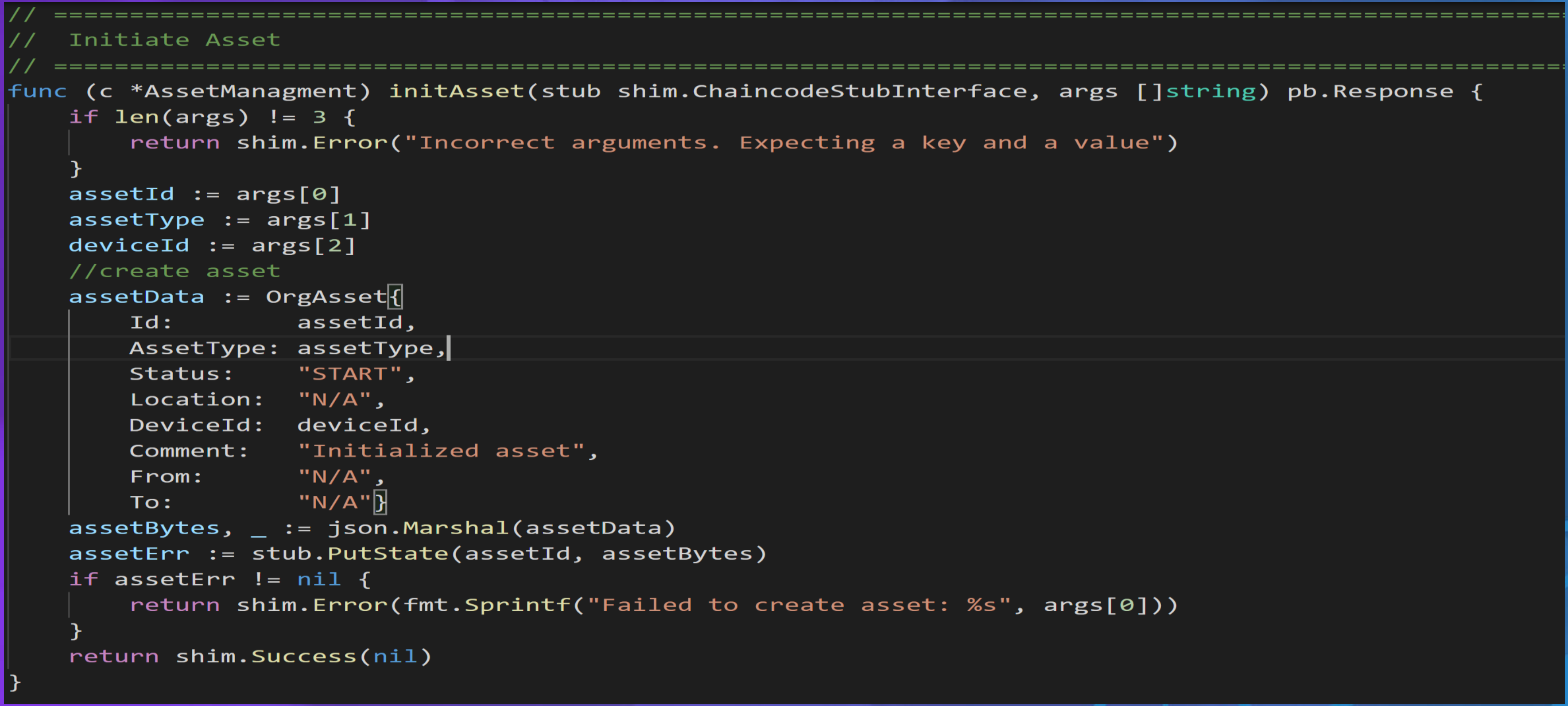
HYPERLEDGER
MEETUP

# Step 4



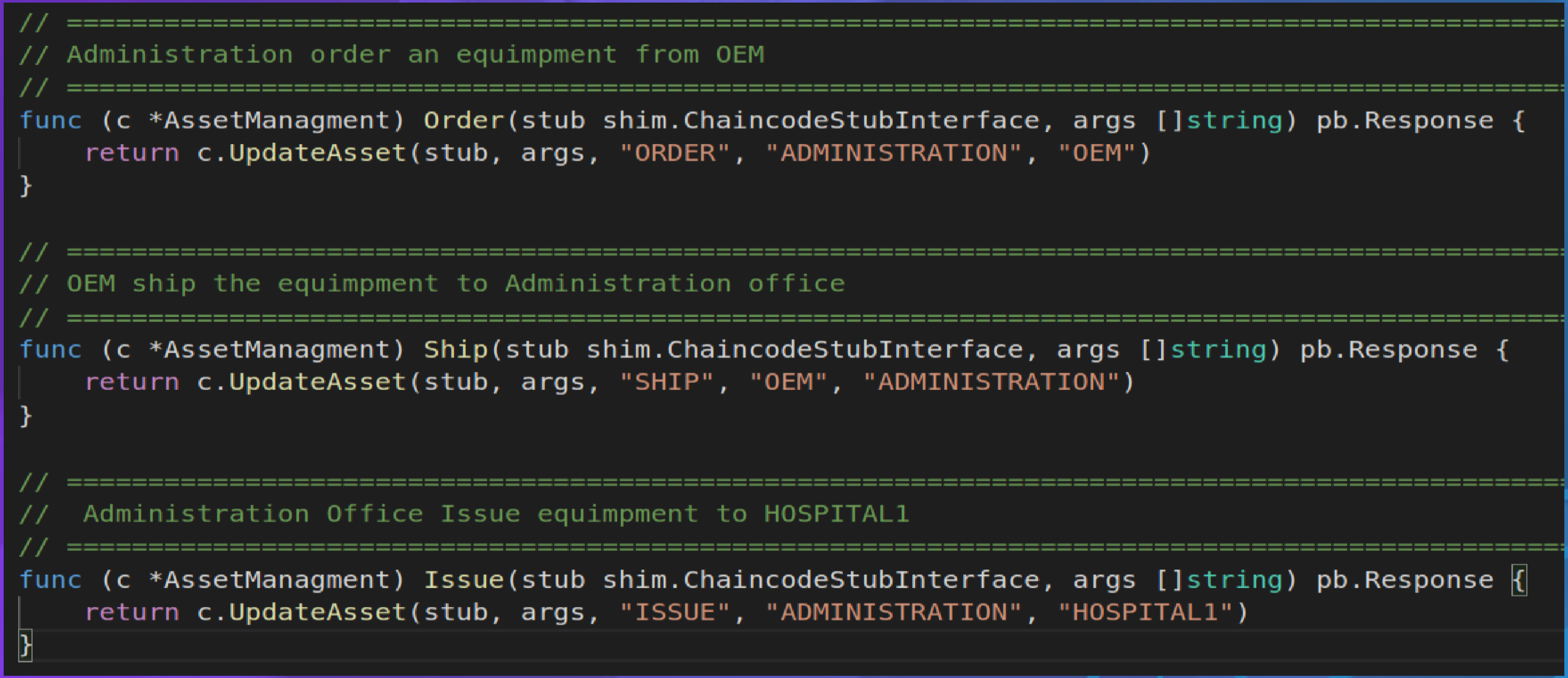
```
// =====================================================================================
// Dynamic Invoke Asset management function
// =====================================================================================
func (c *AssetManagment) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
    function, args := stub.GetFunctionAndParameters()
    if function == "init" {
        return c.initAsset(stub, args)
    }else if function == "Order" {
        return c.Order(stub, args)
    } else if function == "Ship" {
        return c.Ship(stub, args)
    } else if function == "Issue" {
        return c.Issue(stub, args)
    } else if function == "query" {
        return c.query(stub, args)
    } else if function == "getHistory" {
        return c.getHistory(stub, args)
    }

    return shim.Error("Invalid function name")
}
```

# Step 5



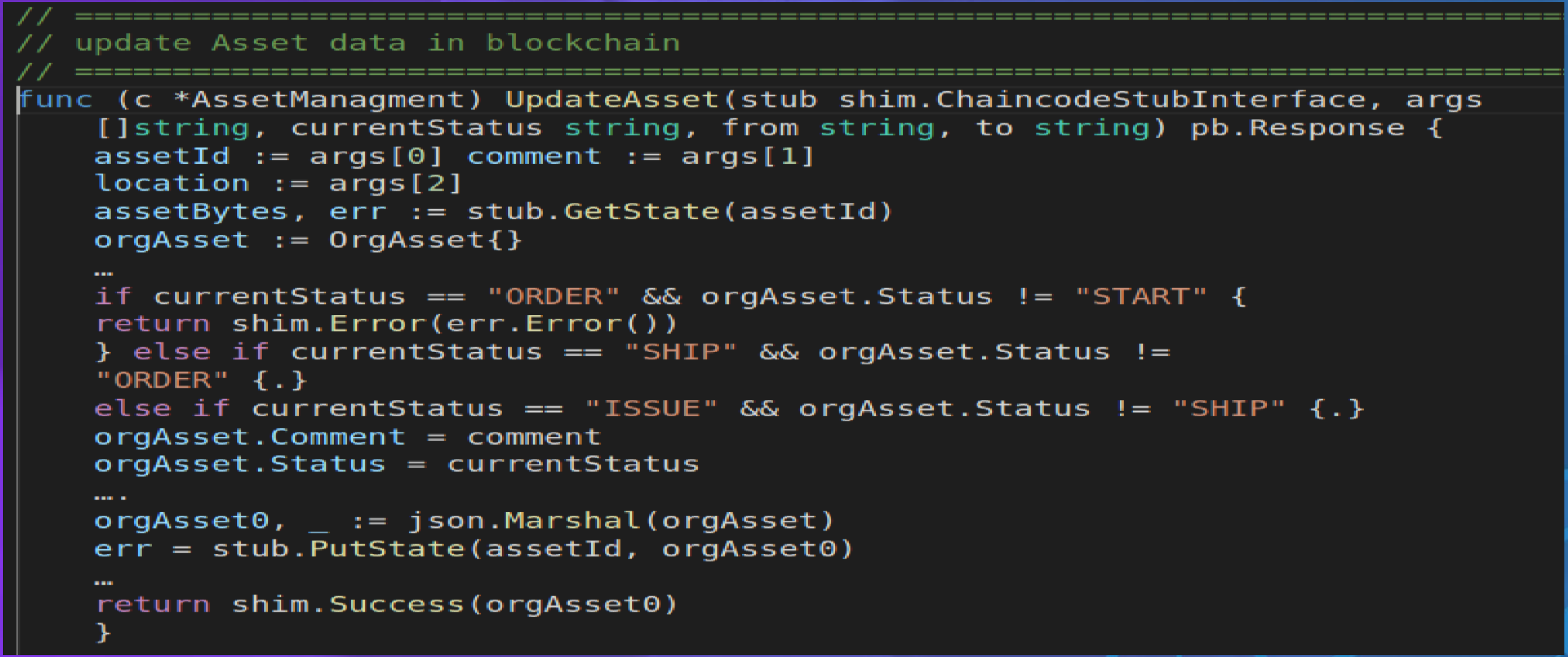
```go
// ===================================================================================================
//   Initiate Asset
// ===================================================================================================
func (c *AssetManagment) initAsset(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    if len(args) != 3 {
        return shim.Error("Incorrect arguments. Expecting a key and a value")
    }
    assetId := args[0]
    assetType := args[1]
    deviceId := args[2]
    //create asset
    assetData := OrgAsset{
        Id:         assetId,
        AssetType: assetType,
        Status:     "START",
        Location:  "N/A",
        DeviceId:  deviceId,
        Comment:    "Initialized asset",
        From:       "N/A",
        To:         "N/A"}
    assetBytes, _ := json.Marshal(assetData)
    assetErr := stub.PutState(assetId, assetBytes)
    if assetErr != nil {
        return shim.Error(fmt.Sprintf("Failed to create asset: %s", args[0]))
    }
    return shim.Success(nil)
}
```

# Step 5 (cont.)



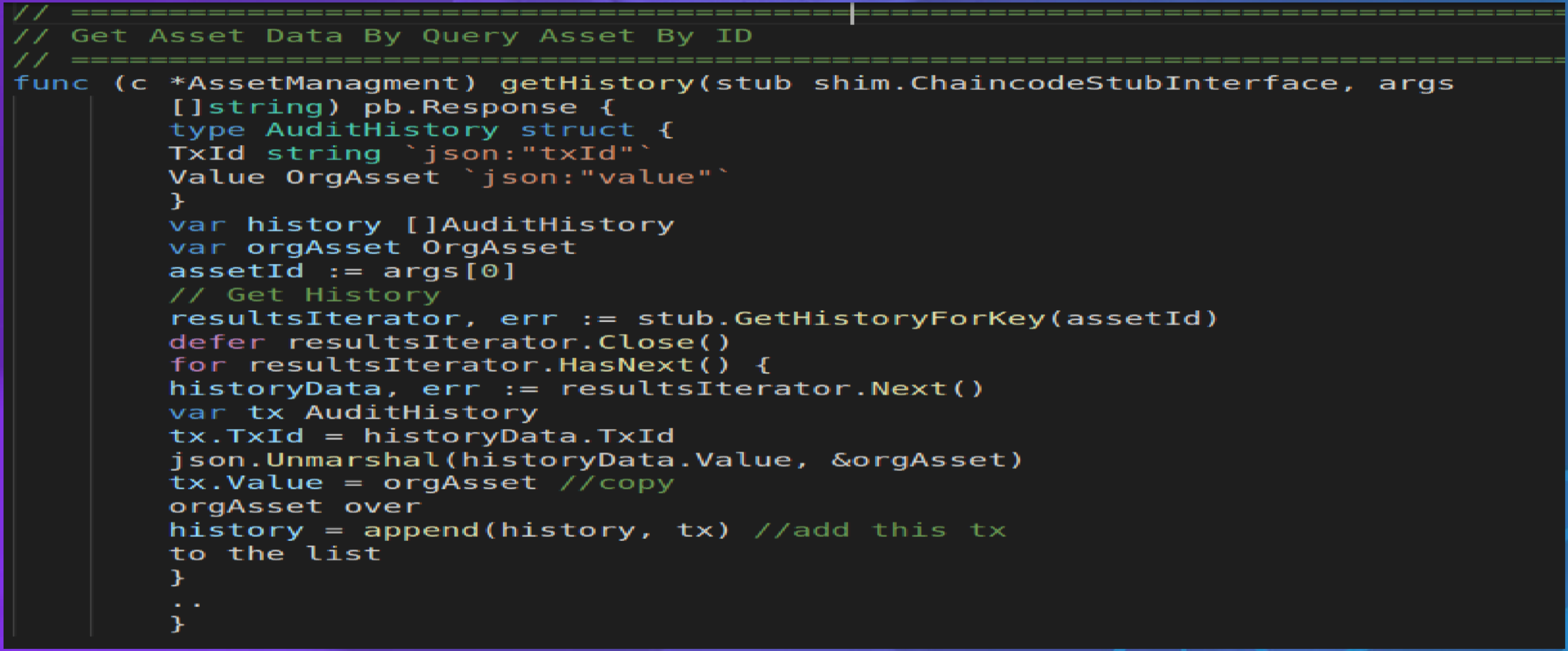
```
// ==========================================================================================
// Administration order an equimpment from OEM
// ==========================================================================================
func (c *AssetManagment) Order(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    return c.UpdateAsset(stub, args, "ORDER", "ADMINISTRATION", "OEM")
}

// ==========================================================================================
// OEM ship the equimpment to Administration office
// ==========================================================================================
func (c *AssetManagment) Ship(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    return c.UpdateAsset(stub, args, "SHIP", "OEM", "ADMINISTRATION")
}

// ==========================================================================================
//  Administration Office Issue equimpment to HOSPITAL1
// ==========================================================================================
func (c *AssetManagment) Issue(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    return c.UpdateAsset(stub, args, "ISSUE", "ADMINISTRATION", "HOSPITAL1")
}
```

# Step 5 (cont.)



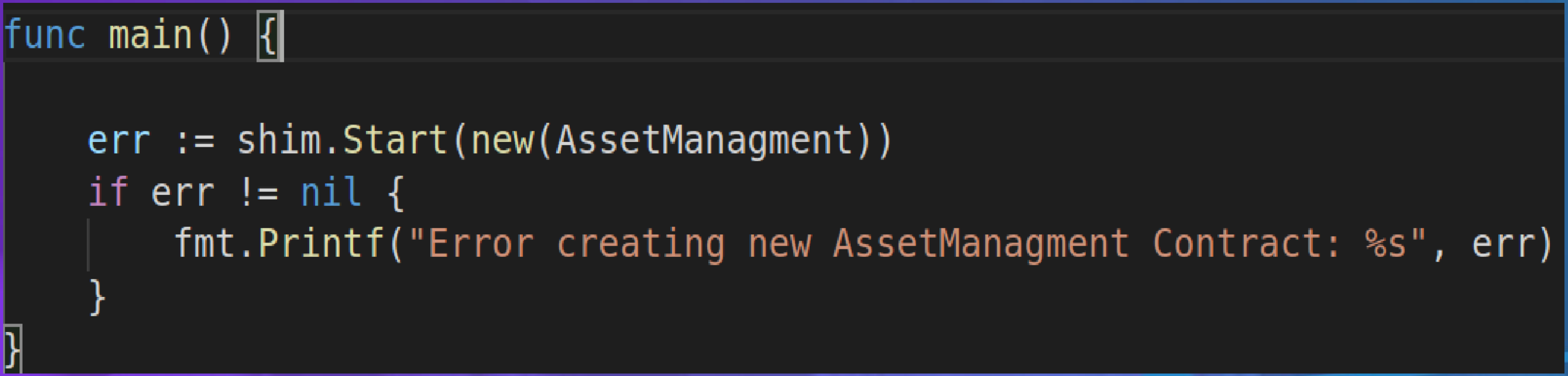
```
// ============================================================================
// update Asset data in blockchain
// ============================================================================
func (c *AssetManagment) UpdateAsset(stub shim.ChaincodeStubInterface, args
    []string, currentStatus string, from string, to string) pb.Response {
    assetId := args[0] comment := args[1]
    location := args[2]
    assetBytes, err := stub.GetState(assetId)
    orgAsset := OrgAsset{}
    …
    if currentStatus == "ORDER" && orgAsset.Status != "START" {
    return shim.Error(err.Error())
    } else if currentStatus == "SHIP" && orgAsset.Status !=
    "ORDER" {.}
    else if currentStatus == "ISSUE" && orgAsset.Status != "SHIP" {.}
    orgAsset.Comment = comment
    orgAsset.Status = currentStatus
    ….
    orgAsset0, _ := json.Marshal(orgAsset)
    err = stub.PutState(assetId, orgAsset0)
    …
    return shim.Success(orgAsset0)
    }
```

# Step 5 (cont.)



```
// ==========================================================================
// Get Asset Data By Query Asset By ID
// ==========================================================================
func (c *AssetManagment) getHistory(stub shim.ChaincodeStubInterface, args
        []string) pb.Response {
        type AuditHistory struct {
        TxId string `json:"txId"`
        Value OrgAsset `json:"value"`
        }
        var history []AuditHistory
        var orgAsset OrgAsset
        assetId := args[0]
        // Get History
        resultsIterator, err := stub.GetHistoryForKey(assetId)
        defer resultsIterator.Close()
        for resultsIterator.HasNext() {
        historyData, err := resultsIterator.Next()
        var tx AuditHistory
        tx.TxId = historyData.TxId
        json.Unmarshal(historyData.Value, &orgAsset)
        tx.Value = orgAsset //copy
        orgAsset over
        history = append(history, tx) //add this tx
        to the list
        }
        ..
        }
```

# Step 6



```go
func main() {

    err := shim.Start(new(AssetManagment))
    if err != nil {
        fmt.Printf("Error creating new AssetManagment Contract: %s", err)
    }
}
```

# Demo