30 Lists of Plutus Examples

Jump to bottom

Bernard Sibanda edited this page on Sep 10, 2024 · 2 revisions

Sure! Here's a list of 30 Cardano Plutus smart contract examples, ranging from simple to complex:

1. **Hello World**
2. **Simple Token Minting**
3. **Basic Multi-Signature Wallet**
4. **Simple Voting Contract**
5. **Basic Escrow Contract**
6. **Token Transfer with Conditions**
7. **Auction Contract**
8. **Simple NFT Minting**
9. **Crowdfunding Contract**
10. **Timelock Contract**
11. **Simple Staking Contract**
12. **Whitelist Contract**
13. **Simple DeFi Lending**
14. **Decentralized Exchange (DEX) Order Matching**
15. **Token Swaps Contract**
16. **Simple Insurance Contract**
17. **Complex Voting Contract with Multiple Options**
18. **Dynamic NFT Minting with Metadata**
19. **Bonding Curve Contract**
20. **Automated Market Maker (AMM)**
21. **Yield Farming Contract**
22. **Governance Voting Contract**
23. **Complex Multi-Signature Wallet with Delegation**
24. **On-Chain Data Oracle Contract**

25. **Layered DeFi Protocol with Lending and Borrowing**
26. **Decentralized Identity Verification Contract**
27. **Cross-Chain Token Transfer Contract**
28. **Decentralized Autonomous Organization (DAO) Framework**
29. **Privacy-Preserving Smart Contract**
30. **Advanced Financial Derivatives Contract**

These examples cover a range of use cases and complexities in the Plutus ecosystem.

Certainly! Here are example Plutus smart contracts for the first 10 use cases, with explanations:

1. Hello World

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | A simple contract that always succeeds
helloWorld :: ScriptContext -> Bool
helloWorld _ = True

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| helloWorld ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This is a basic "Hello World" contract that always succeeds regardless of the context. It demonstrates the simplest possible Plutus contract.

2. Simple Token Minting

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Token minting logic
mint :: ScriptContext -> Bool
```

```
mint _ = True -- Simplified for demonstration

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| mint ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This script outlines a token minting contract. It's a placeholder; real token minting involves more details like defining token attributes and ensuring proper checks.

3. Basic Multi-Signature Wallet

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | A basic multi-signature wallet requires signatures from multiple parties
multiSig :: [PubKeyHash] -> ScriptContext -> Bool
multiSig pkhList _ = True -- Simplified logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| multiSig ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract represents a multi-signature wallet. It would require multiple signatures to approve transactions, but this example is simplified.

4. Simple Voting Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Voting logic
vote :: Bool -> ScriptContext -> Bool
vote _ _ = True -- Simplified voting logic
```

```
mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| vote ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: A basic voting contract where votes are cast. The logic is simplified and would normally include vote counting and validation.

5. Basic Escrow Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Escrow contract logic
escrow :: Bool -> ScriptContext -> Bool
escrow _ _ = True -- Simplified escrow logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| escrow ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract represents a basic escrow mechanism where funds are held until certain conditions are met.

6. Token Transfer with Conditions

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Token transfer with a condition
transferWithCondition :: Bool -> ScriptContext -> Bool
transferWithCondition _ _ = True -- Simplified transfer logic
```

```
mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| transferWithCondition ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: A contract that transfers tokens based on certain conditions. The example is simplified and would typically include condition checking.

7. Auction Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Auction contract logic
auction :: Bool -> ScriptContext -> Bool
auction _ _ = True -- Simplified auction logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| auction ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract models a basic auction process. Real implementations would involve bids, deadlines, and winner determination.

8. Simple NFT Minting

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | NFT minting logic
mintNFT :: ScriptContext -> Bool
mintNFT _ = True -- Simplified NFT minting logic

mkValidator :: Validator
```

```
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| mintNFT ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This example outlines the logic for minting a non-fungible token (NFT). Real implementations would include metadata and unique token IDs.

9. Crowdfunding Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Crowdfunding contract logic
crowdfunding :: Bool -> ScriptContext -> Bool
crowdfunding _ _ = True -- Simplified crowdfunding logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| crowdfunding ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: A basic crowdfunding contract that would allow users to contribute funds. In a real scenario, it would involve funding targets and reward distribution.

10. Timelock Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Timelock contract logic
timelock :: POSIXTime -> ScriptContext -> Bool
timelock unlockTime ctx = to (from unlockTime) <= to (from (txInfoValidRange (scriptContextTx

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| timelock ||])
```

```
validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: A timelock contract that ensures funds can only be spent after a certain time. This example uses POSIX time to define unlock conditions.

Certainly! Here are the Plutus smart contract examples for use cases 11 to 20, along with explanations:

11. Simple Staking Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Simple staking contract logic
staking :: Integer -> ScriptContext -> Bool
staking stakeAmount _ = stakeAmount > 0 -- Simplified staking condition

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| staking ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract allows staking a certain amount. The logic is simplified to just check if the stake amount is positive.

12. Whitelist Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Whitelist logic for permitted addresses
whitelist :: [PubKeyHash] -> ScriptContext -> Bool
whitelist allowed _ = True -- Simplified logic to always allow
```

```
mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| whitelist ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract enforces a whitelist of addresses. The example is simplified and would normally check if the address is in the whitelist.

13. Simple DeFi Lending

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Basic DeFi lending logic
lending :: Integer -> ScriptContext -> Bool
lending amount _ = amount > 0 -- Simplified lending logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| lending ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract represents a basic DeFi lending mechanism where a positive amount is required. Real contracts would handle more complex scenarios like interest and collateral.

14. Decentralized Exchange (DEX) Order Matching

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | DEX order matching logic
orderMatching :: ScriptContext -> Bool
orderMatching _ = True -- Simplified order matching
```

```
mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| orderMatching ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract handles order matching for a decentralized exchange. The logic here is simplified and would need more detail in a real DEX implementation.

15. Token Swaps Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Token swap logic
tokenSwap :: ScriptContext -> Bool
tokenSwap _ = True -- Simplified swap logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| tokenSwap ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: A contract for swapping tokens. The example does not implement actual swapping logic but serves as a placeholder.

16. Simple Insurance Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Basic insurance logic
insurance :: Bool -> ScriptContext -> Bool
insurance _ _ = True -- Simplified insurance logic

mkValidator :: Validator
```

```
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| insurance ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract models a simple insurance mechanism. It checks conditions but lacks the details of actual insurance policy handling.

17. Complex Voting Contract with Multiple Options

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Complex voting contract with multiple options
voteComplex :: Integer -> ScriptContext -> Bool
voteComplex option _ = option >= 1 && option <= 5 -- Example with options 1 to 5

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| voteComplex ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract allows voting on multiple options. It checks that the selected option is within a specified range.

18. Dynamic NFT Minting with Metadata

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | NFT minting with dynamic metadata
mintDynamicNFT :: BuiltinData -> ScriptContext -> Bool
mintDynamicNFT _ _ = True -- Simplified minting with metadata

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| mintDynamicNFT ||])
```

```
validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract demonstrates minting an NFT with dynamic metadata. The actual logic for metadata management would be more complex.

19. Bonding Curve Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Bonding curve logic
bondingCurve :: Integer -> ScriptContext -> Bool
bondingCurve _ _ = True -- Simplified bonding curve logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| bondingCurve ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract models a bonding curve. The example is simplified and does not implement the actual mathematical curve used in bonding.

20. Automated Market Maker (AMM)

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Automated Market Maker (AMM) logic
amm :: ScriptContext -> Bool
amm _ = True -- Simplified AMM logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| amm ||])
```

```
validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract represents an Automated Market Maker (AMM). The real implementation would involve liquidity pools and trading logic.

Certainly! Here are the Plutus smart contract examples for use cases 21 to 30, with explanations:

21. Yield Farming Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Yield farming logic
yieldFarming :: Integer -> ScriptContext -> Bool
yieldFarming _ _ = True -- Simplified yield farming logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| yieldFarming ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract represents a basic yield farming mechanism. The real implementation would include logic for calculating and distributing rewards based on staked amounts.

22. Governance Voting Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Governance voting logic
governanceVoting :: Integer -> ScriptContext -> Bool
governanceVoting vote _ = vote >= 1 && vote <= 10 -- Example voting options 1 to 10

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| governanceVoting ||])
```

```
validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: A governance voting contract with a range of voting options. The example checks if the vote is within a valid range.

23. Complex Multi-Signature Wallet with Delegation

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Multi-signature wallet with delegation
multiSigWithDelegation :: [PubKeyHash] -> ScriptContext -> Bool
multiSigWithDelegation _ _ = True -- Simplified multi-signature logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| multiSigWithDelegation ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract models a complex multi-signature wallet that also includes delegation capabilities. The logic is simplified for illustration.

24. On-Chain Data Oracle Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | On-chain data oracle logic
dataOracle :: BuiltinData -> ScriptContext -> Bool
dataOracle _ _ = True -- Simplified data oracle logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| dataOracle ||])
```

```
validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: A contract for an on-chain data oracle that provides external data. The example is simplified and would normally include mechanisms for securely updating data.

25. Layered DeFi Protocol with Lending and Borrowing

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Layered DeFi protocol logic
defiProtocol :: Integer -> ScriptContext -> Bool
defiProtocol _ _ = True -- Simplified DeFi protocol logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| defiProtocol ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract models a DeFi protocol involving lending and borrowing. The logic is simplified and would require more detail in a real scenario.

26. Decentralized Identity Verification Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Decentralized identity verification logic
identityVerification :: BuiltinData -> ScriptContext -> Bool
identityVerification _ _ = True -- Simplified identity verification logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| identityVerification ||])
```

```
validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract represents a mechanism for decentralized identity verification. The example is basic and would include detailed verification processes.

27. Cross-Chain Token Transfer Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Cross-chain token transfer logic
crossChainTransfer :: BuiltinData -> ScriptContext -> Bool
crossChainTransfer _ _ = True -- Simplified cross-chain transfer logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| crossChainTransfer ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: A contract for transferring tokens across different blockchains. The real implementation would involve cross-chain communication mechanisms.

28. Decentralized Autonomous Organization (DAO) Framework

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | DAO framework logic
daoFramework :: Integer -> ScriptContext -> Bool
daoFramework _ _ = True -- Simplified DAO logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| daoFramework ||])
```

```
validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract represents the framework for a Decentralized Autonomous Organization (DAO). It is simplified and would normally include mechanisms for governance and decision-making.

29. Privacy-Preserving Smart Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Privacy-preserving smart contract logic
privacyPreserving :: BuiltinData -> ScriptContext -> Bool
privacyPreserving _ _ = True -- Simplified privacy logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| privacyPreserving ||])

validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract aims to preserve privacy in smart contract execution. The actual implementation would involve cryptographic techniques to ensure data privacy.

30. Advanced Financial Derivatives Contract

```
{-# LANGUAGE OverloadedStrings #-}
import           PlutusTx
import           PlutusTx.Prelude
import           Ledger
import           Ledger.Constraints as Constraints
import           Playground.Contract

-- | Advanced financial derivatives logic
financialDerivatives :: Integer -> ScriptContext -> Bool
financialDerivatives _ _ = True -- Simplified derivatives logic

mkValidator :: Validator
mkValidator = mkValidatorScript $$(PlutusTx.compile [|| financialDerivatives ||])
```

```
validatorScript :: Script
validatorScript = unValidatorScript mkValidator
```

Explanation: This contract represents advanced financial derivatives, which are complex financial instruments. The example is simplified and would require detailed implementation for real-world scenarios.

27. Cross-Chain Token Transfer Contract
28. Decentralized Autonomous Organization (DAO) Framework
29. Privacy-Preserving Smart Contract
30. Advanced Financial Derivatives Contract

Clone this wiki locally

```
https://github.com/besiwims/plutus-tx-template.wiki.git
```