Date: 13-01-2022
Student: Rob Honig

Course: CP108 Plutus/Haskell I

Problem statement: Most African governments are trying different measures to eliminate *corruption*, *bad governance*, *mismanagement* and *lack of accountability* in their countries but these efforts are mostly unsuccessful. As a tool for change, *blockchain* can help in solving some of these governance issues plaguing Africa.

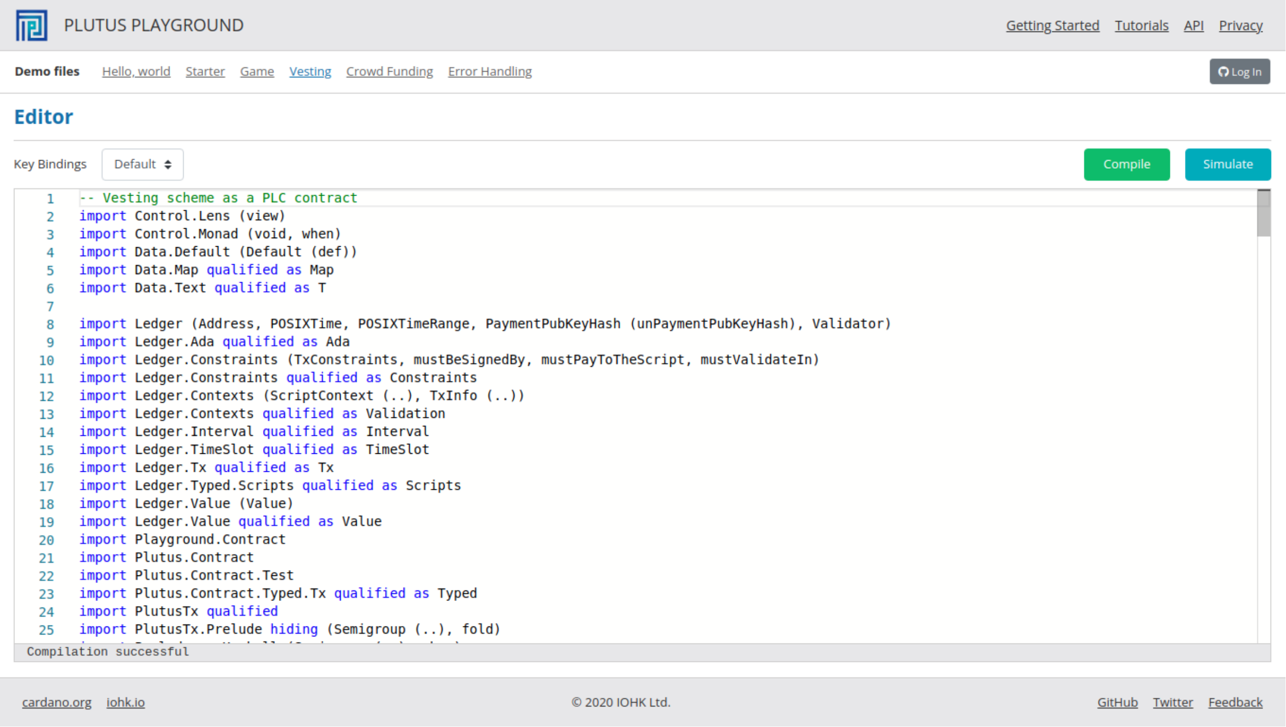
Task: With this in mind, think of a way you can implement a smart contract that can help eliminate any of these ills, then implement it using Plutus.

Possible solution: Giving acces to funds without using the so called “Middleman”. We use “Smart Contracts” for (financial) aid or microcredits, powered by the Cardano Blockchain. Lock Ada funds on the blockchain, redeem funds with the correct redeemer.

In this way no intermediary is used. This keeps human error to a minimum. Based on predetermined rules written in Haskell and Plutus.

- No middlemen involved;
- Low transaction costs;
- 24/7 available;
- Safe, secure and fast

Script: The ***Gift.hs*** can be used for this exam and can be tested at the Plutus Playground;
https://playground.plutus.iohkdev.io/



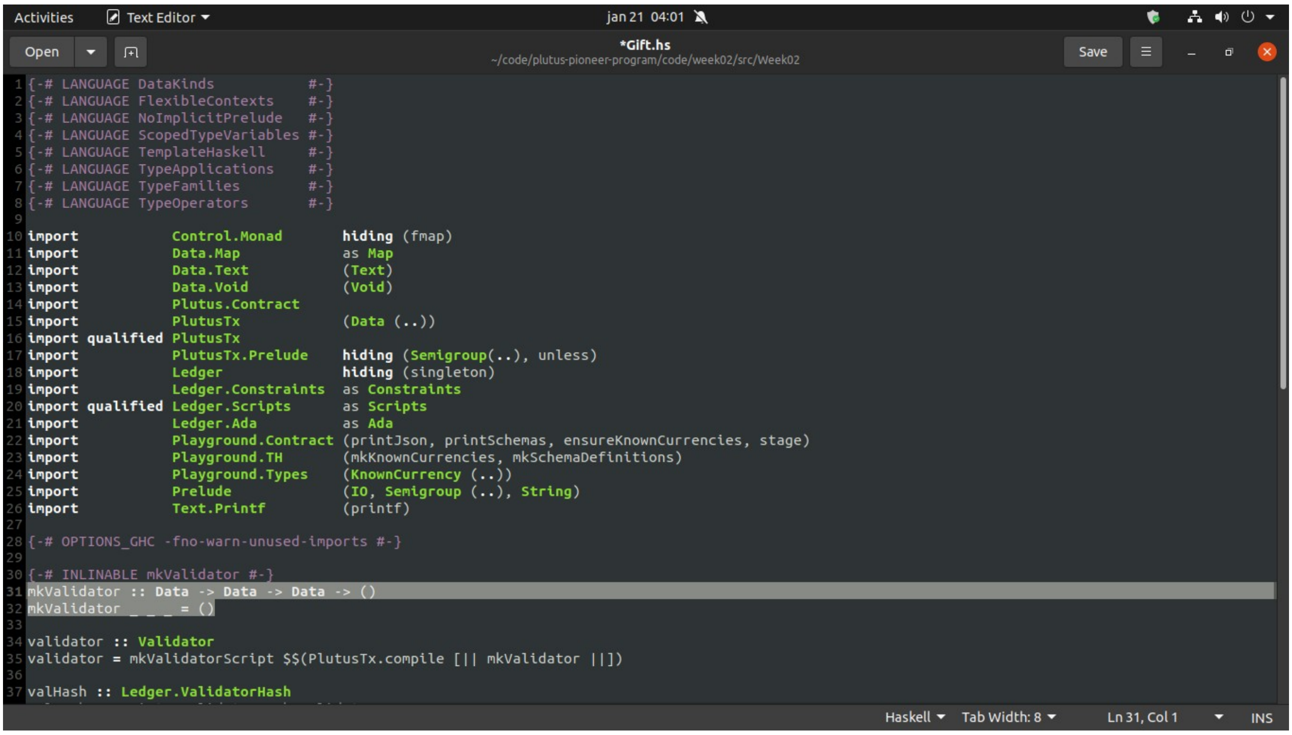
```haskell
-- Vesting scheme as a PLC contract
import Control.Lens (view)
import Control.Monad (void, when)
import Data.Default (Default (def))
import Data.Map qualified as Map
import Data.Text qualified as T

import Ledger (Address, POSIXTime, POSIXTimeRange, PaymentPubKeyHash (unPaymentPubKeyHash), Validator)
import Ledger.Ada qualified as Ada
import Ledger.Constraints (TxConstraints, mustBeSignedBy, mustPayToTheScript, mustValidateIn)
import Ledger.Constraints qualified as Constraints
import Ledger.Contexts (ScriptContext (..), TxInfo (..))
import Ledger.Contexts qualified as Validation
import Ledger.Interval qualified as Interval
import Ledger.TimeSlot qualified as TimeSlot
import Ledger.Tx qualified as Tx
import Ledger.Typed.Scripts qualified as Scripts
import Ledger.Value (Value)
import Ledger.Value qualified as Value
import Playground.Contract
import Plutus.Contract
import Plutus.Contract.Test
import Plutus.Contract.Typed.Tx qualified as Typed
import PlutusTx qualified
import PlutusTx.Prelude hiding (Semigroup (..), fold)
```

```
Compilation successful
```

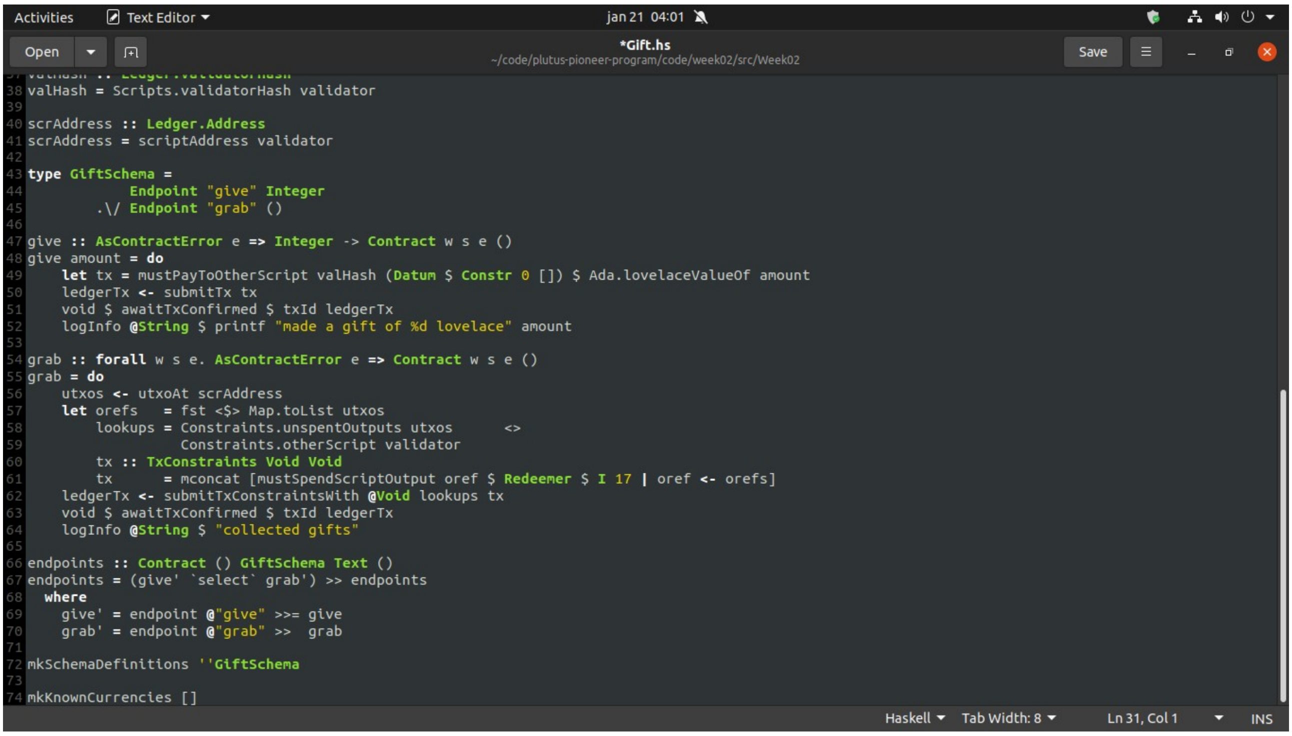
Language: Haskell and Plutus

Idea: The idea is to create a marketplace where lenders and borrowers can meet to take out micro loans. As a result, borrowers can obtain funds without the intervention of an intermediary. After both parties have reached an agreement, the lender can prepare a loan that can be redeemed by the borrower using a code. To accomplish this we can use the previously mentioned ***Gift.hs*** script example from Week 02 of the Plutus Pioneer Program.

Example *Gift.hs*:



```haskell
{-# LANGUAGE DataKinds           #-}
{-# LANGUAGE FlexibleContexts    #-}
{-# LANGUAGE NoImplicitPrelude   #-}
{-# LANGUAGE ScopedTypeVariables #-}
{-# LANGUAGE TemplateHaskell     #-}
{-# LANGUAGE TypeApplications    #-}
{-# LANGUAGE TypeFamilies        #-}
{-# LANGUAGE TypeOperators       #-}

import           Control.Monad       hiding (fmap)
import           Data.Map            as Map
import           Data.Text           (Text)
import           Data.Void           (Void)
import           Plutus.Contract
import           PlutusTx            (Data (..))
import qualified PlutusTx
import           PlutusTx.Prelude    hiding (Semigroup(..), unless)
import           Ledger              hiding (singleton)
import           Ledger.Constraints  as Constraints
import qualified Ledger.Scripts      as Scripts
import           Ledger.Ada          as Ada
import           Playground.Contract (printJson, printSchemas, ensureKnownCurrencies, stage)
import           Playground.TH       (mkKnownCurrencies, mkSchemaDefinitions)
import           Playground.Types    (KnownCurrency (..))
import           Prelude             (IO, Semigroup (..), String)
import           Text.Printf         (printf)

{-# OPTIONS_GHC -fno-warn-unused-imports #-}

{-# INLINABLE mkValidator #-}
mkValidator :: Data -> Data -> Data -> ()
mkValidator _ _ _ = ()

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

valHash :: Ledger.ValidatorHash
```



```haskell
valHash = Scripts.validatorHash validator

scrAddress :: Ledger.Address
scrAddress = scriptAddress validator

type GiftSchema =
            Endpoint "give" Integer
        .\/ Endpoint "grab" ()

give :: AsContractError e => Integer -> Contract w s e ()
give amount = do
    let tx = mustPayToOtherScript valHash (Datum $ Constr 0 []) $ Ada.lovelaceValueOf amount
    ledgerTx <- submitTx tx
    void $ awaitTxConfirmed $ txId ledgerTx
    logInfo @String $ printf "made a gift of %d lovelace" amount

grab :: forall w s e. AsContractError e => Contract w s e ()
grab = do
    utxos <- utxoAt scrAddress
    let orefs   = fst <$> Map.toList utxos
        lookups = Constraints.unspentOutputs utxos      <>
                  Constraints.otherScript validator
        tx :: TxConstraints Void Void
        tx      = mconcat [mustSpendScriptOutput oref $ Redeemer $ I 17 | oref <- orefs]
    ledgerTx <- submitTxConstraintsWith @Void lookups tx
    void $ awaitTxConfirmed $ txId ledgerTx
    logInfo @String $ "collected gifts"

endpoints :: Contract () GiftSchema Text ()
endpoints = (give' `select` grab') >> endpoints
  where
    give' = endpoint @"give" >>= give
    grab' = endpoint @"grab" >>  grab

mkSchemaDefinitions ''GiftSchema

mkKnownCurrencies []
```

Code: Important part for us but also of this script is validation. The borrower gets his personal private key / code that he can use to redeem funds. He will be able to unlock the funds when valid.

The ***Gift.hs*** script can be integrated in the marketplace. The Haskell function mkValidator (1) that represents the validator can be modified. The function consists of three arguments: Datum, Redeemer and ScriptContext. These share the same data types called data.

(1) mkValidator :: Data → Data → Data → ()

(2) mkValidator :: () → Integer → ScriptContext → Bool

The modfied mkValidator (2) is an example. The lender can give the borrower the code to unlock the funds in the Marketplace. In this example the borrowers code is represented by an Integer.

Custom data types can be used too and we did not talk about the ScriptContext. There are lots of possibilities with the validation script to transfer or donate funds without the use of a Middleman and this keeps human error to a minimum.