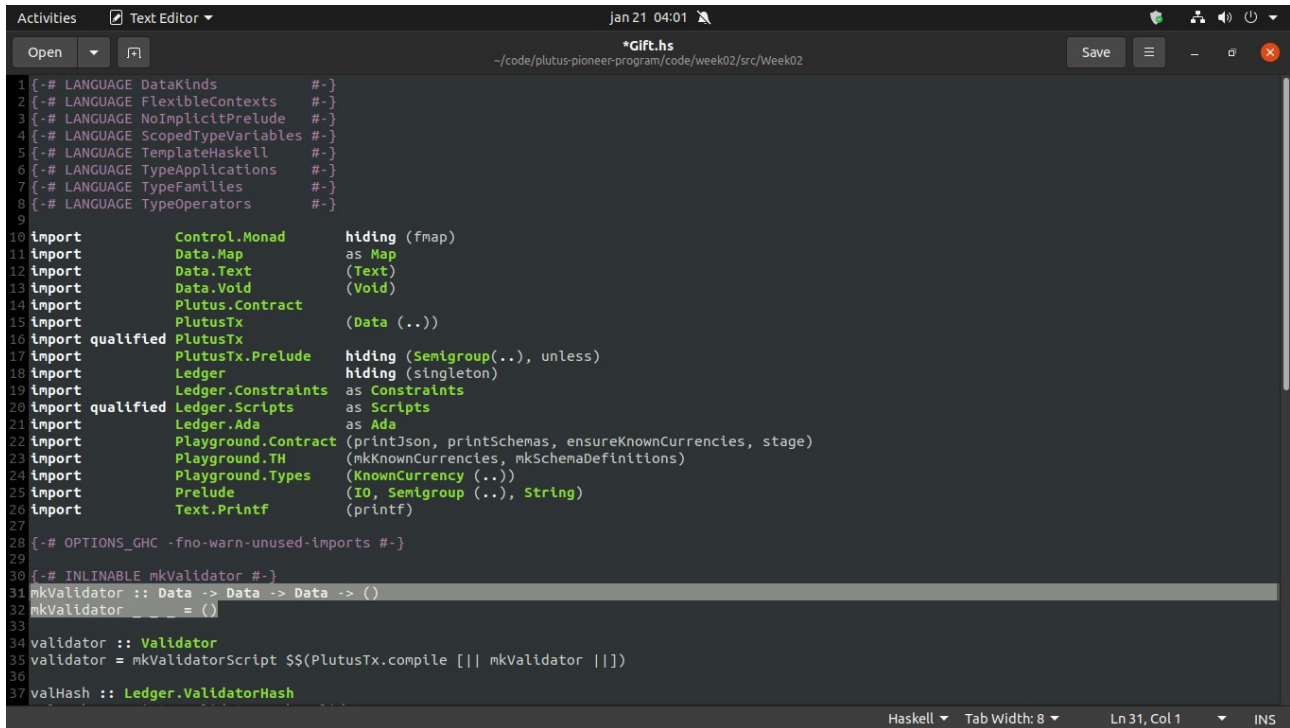
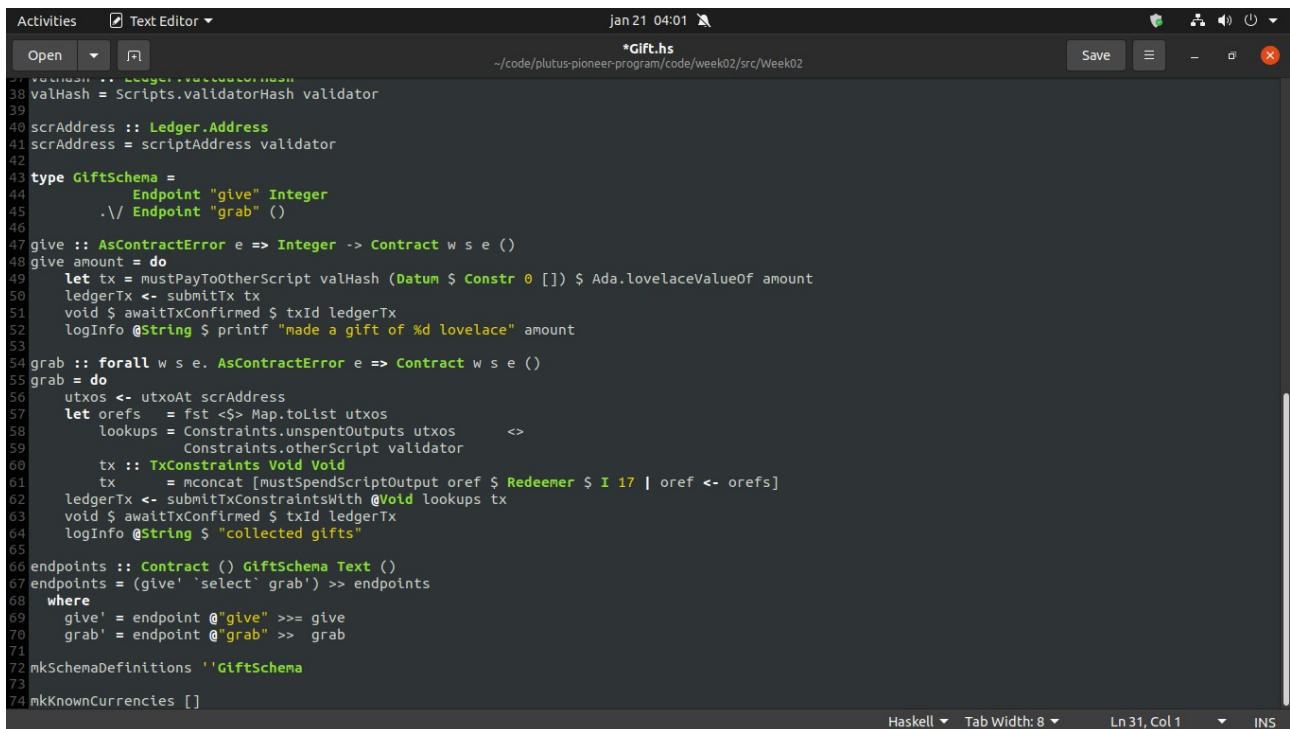


**Idea:** The idea is to create a marketplace where lenders and borrowers can meet to take out micro loans. As a result, borrowers can obtain funds without the intervention of an intermediary. After both parties have reached an agreement, the lender can prepare a loan that can be redeemed by the borrower 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*:



```
1 {-# LANGUAGE DataKinds           #-}
2 {-# LANGUAGE FlexibleContexts     #-}
3 {-# LANGUAGE NoImplicitPrelude    #-}
4 {-# LANGUAGE ScopedTypeVariables #-}
5 {-# LANGUAGE TemplateHaskell     #-}
6 {-# LANGUAGE TypeApplications    #-}
7 {-# LANGUAGE TypeFamilies        #-}
8 {-# LANGUAGE TypeOperators       #-}
9
10 import           Control.Monad      hiding (fmap)
11 import           Data.Map           as Map
12 import           Data.Text          (Text)
13 import           Data.Void          (Void)
14 import           Plutus.Contract
15 import           PlutusTx           (Data (...))
16 import qualified PlutusTx           as PlutusTx
17 import           PlutusTx.Prelude   hiding (Semigroup(..), unless)
18 import           Ledger             hiding (singleton)
19 import           Ledger.Constraints as Constraints
20 import qualified Ledger.Scripts      as Scripts
21 import           Ledger.Ada         as Ada
22 import           Playground.Contract (printJson, printSchemas, ensureKnownCurrencies, stage)
23 import           Playground.TH      (mkKnownCurrencies, mkSchemaDefinitions)
24 import           Playground.Types   (KnownCurrency (...))
25 import           Prelude            (IO, Semigroup (...), String)
26 import           Text.Printf        (printf)
27
28 {-# OPTIONS_GHC -fno-warn-unused-imports #-}
29
30 {-# INLINABLE mkValidator #-}
31 mkValidator :: Data -> Data -> Data -> ()
32 mkValidator _ _ _ = ()
33
34 validator :: Validator
35 validator = mkValidatorScript $(PlutusTx.compile [] mkValidator [])
36
37 valHash :: Ledger.ValidatorHash
```



```
38 valHash = Scripts.validatorHash validator
39
40 scrAddress :: Ledger.Address
41 scrAddress = scriptAddress validator
42
43 type GiftSchema =
44   Endpoint "give" Integer
45   .\ Endpoint "grab" ()
46
47 give :: AsContractError e => Integer -> Contract w s e ()
48 give amount = do
49   let tx = mustPayToOtherScript valHash (Datum $ Constr 0 []) $ Ada.lovelaceValueOf amount
50   ledgerTx <- submitTx tx
51   void $ awaitTxConfirmed $ txId ledgerTx
52   logInfo @String $ printf "made a gift of %d lovelace" amount
53
54 grab :: forall w s e. AsContractError e => Contract w s e ()
55 grab = do
56   utxos <- utxoAt scrAddress
57   let orefs = fst <$> Map.toList utxos
58       lookups = Constraints.unspentOutputs utxos <=>
59               Constraints.otherScript validator
60   tx :: TxConstraints Void Void
61   tx = mconcat [mustSpendScriptOutput oref $ Redeemer $ I 17 | oref <- orefs]
62   ledgerTx <- submitTxConstraintsWith @Void lookups tx
63   void $ awaitTxConfirmed $ txId ledgerTx
64   logInfo @String $ "collected gifts"
65
66 endpoints :: Contract () GiftSchema Text ()
67 endpoints = (give `select` grab) >> endpoints
68 where
69   give' = endpoint @"give" >=> give
70   grab' = endpoint @"grab" >> grab
71
72 mkSchemaDefinitions ''GiftSchema
73
74 mkKnownCurrencies []
```