Some code template

Some shell code.

```
Prelude> import Vesting
Prelude Vesting> :set -XOverloadedStrings
Prelude Vesting> import Plutus.V2.Ledger.Api
Prelude Vesting Plutus.V1.Ledger.Api>
pkh1 = "cff6e39ec5b3cf84b1078976c98706b73774d2c5523af4daaf7c5109"
Prelude Vesting Plutus.V1.Ledger.Api>
printVestingDatumJSON pkh1 "2023-03-11T13:12:11.123Z"
{
    "constructor": 0,
    "fields": [
            {
                  "bytes": "cff6e39ec5b3cf84b1078976c98706b73774d2c5523af4daaf7c5109"
            },
            {
                  "int": 1678540331123
            }
    ]
}
```

Some shell code for a Marlowe contract.

```
When
    [Case
        (Deposit
            (Role "Alice")
            (Role "Alice")
            (Token "" "")
            (Constant 10)
        )
        (When
            [Case
                (Deposit
                    (Role "Bob")
                    (Role "Bob")
                    (Token "" "")
                    (Constant 10)
                )
                (When
                    [Case
                        (Choice
                            (ChoiceId
                                "Winner"
                                (Role "Charlie")
                            )
```

```
                            [Bound 1 2]
                        )
                        (If
                            (ValueEQ
                                (ChoiceValue
                                    (ChoiceId
                                        "Winner"
                                        (Role "Charlie")
                                    ))
                                (Constant 1)
                            )
                            (Pay
                                (Role "Bob")
                                (Account (Role "Alice"))
                                (Token "" "")
                                (Constant 10)
                                Close
                            )
                            (Pay
                                (Role "Alice")
                                (Account (Role "Bob"))
                                (Token "" "")
                                (Constant 10)
                                Close
                            )
                        )]
                    1682551111000 Close
                )]
            1682552111000 Close
        )]
    1682553111000 Close
```

Two haskell types.

```
data Data
    = Constr Integer [Data]
    | Map [(Data, Data)]
    | List [Data]
    | I Integer
    | B BS.ByteString
    deriving stock (Show, Read, Eq, Ord, Generic, Data.Data.Data)

newtype Value = Value
    { getValue :: Map.Map CurrencySymbol (Map.Map TokenName Integer)
    }
    deriving stock (Generic, Typeable, Haskell.Show)
```

Some Haskell code.

```
{-# LANGUAGE DataKinds           #-}
{-# LANGUAGE ImportQualifiedPost #-}
{-# LANGUAGE NoImplicitPrelude   #-}
{-# LANGUAGE TemplateHaskell     #-}

module Gift where

import qualified Plutus.V2.Ledger.Api as PlutusV2
import           PlutusTx             (BuiltinData, compile)
import           Prelude              (IO)
import           Utilities            (writeValidatorToFile)

------------------------------------------------------------------------------
-------------------------- ON-CHAIN CODE / VALIDATOR -------------------------

-- This validator always succeeds
--                    Datum         Redeemer     ScriptContext
mkGiftValidator :: BuiltinData -> BuiltinData -> BuiltinData -> ()
mkGiftValidator _ _ _ = ()
{-# INLINABLE mkGiftValidator #-}

validator :: PlutusV2.Validator
validator = PlutusV2.mkValidatorScript $$(PlutusTx.compile
                                        [|| mkGiftValidator ||])


------------------------------------------------------------------------------
------------------------------- HELPER FUNCTIONS -----------------------------

saveVal :: IO ()
saveVal = writeValidatorToFile "./gift.plutus" validator
```

I'm refering to the mkGiftValidator function and the BuiltinData data type.

Some typescript code.

```
import {
    Data,
    Lucid,
    Blockfrost,
} from "https://deno.land/x/lucid@0.9.1/mod.ts"
// create a seed.ts file with your seed
import { secretSeed } from "./seed.ts"

// set blockfrost endpoint
const lucid = await Lucid.new(
  new Blockfrost(
    "https://cardano-preview.blockfrost.io/api/v0",
    "insert your own api key here"
```

```
  ),
  "Preview"
);

// load local stored seed as a wallet into lucid
lucid.selectWalletFromSeed(secretSeed);
const addr: Address = await lucid.wallet.address();
console.log(addr);

// An asynchronous function that sends an amount of Lovelace to the script
// with the above datum.
async function vestFunds(amount: bigint): Promise<TxHash> {
    const dtm: Datum = Data.to<VestingDatum>(datum,VestingDatum);
    const tx = await lucid
      .newTx()
      .payToContract(vestingAddress, { inline: dtm }, { lovelace: amount })
      .complete();
    const signedTx = await tx.sign().complete();
    const txHash = await signedTx.submit();
    return txHash
}

console.log(await vestFunds(100000000n));
```