# PyChain Ledger

################################################################################

# You’ll make the following updates to the provided Python file for this

# Challenge, which already contains the basic `PyChain` ledger structure that

# you created throughout the module:

# Step 1: Create a Record Data Class

# \* Create a new data class named `Record`. This class will serve as the

# blueprint for the financial transaction records that the blocks of the ledger

# will store.

# Step 2: Modify the Existing Block Data Class to Store Record Data

# \* Change the existing `Block` data class by replacing the generic `data`

# attribute with a `record` attribute that’s of type `Record`.

# Step 3: Add Relevant User Inputs to the Streamlit Interface

# \* Create additional user input areas in the Streamlit application. These

# input areas should collect the relevant information for each financial record

# that you’ll store in the `PyChain` ledger.

# Step 4: Test the PyChain Ledger by Storing Records

# \* Test your complete `PyChain` ledger.

################################################################################

# Imports

import streamlit as st

from dataclasses import dataclass

from typing import Any, List

import datetime as datetime

import pandas as pd

import hashlib

################################################################################

# Step 1:

# Create a Record Data Class

# Define a new Python data class named `Record`. Give this new class a

# formalized data structure that consists of the `sender`, `receiver`, and

# `amount` attributes. To do so, complete the following steps:

# 1. Define a new class named `Record`.

# 2. Add the `@dataclass` decorator immediately before the `Record` class

# definition.

# 3. Add an attribute named `sender` of type `str`.

# 4. Add an attribute named `receiver` of type `str`.

# 5. Add an attribute named `amount` of type `float`.

# Note that you’ll use this new `Record` class as the data type of your `record` attribute in the next section.

# @TODO

# Create a Record Data Class that consists of the `sender`, `receiver`, and

# `amount` attributes

@dataclass

class Record:

    sender: str

    receiver: str

    amount: float

################################################################################

# Step 2:

# Modify the Existing Block Data Class to Store Record Data

# Rename the `data` attribute in your `Block` class to `record`, and then set

# it to use an instance of the new `Record` class that you created in the

# previous section. To do so, complete the following steps:

# 1. In the `Block` class, rename the `data` attribute to `record`.

# 2. Set the data type of the `record` attribute to `Record`.

@dataclass

class Block:

    # @TODO

    # Rename the `data` attribute to `record`, and set the data type to `Record`

    # data: Any

    record: Record

   # using the Record class defined in Step 1

    creator\_id: int

    prev\_hash: str = "0"

    timestamp: str = datetime.datetime.utcnow().strftime("%H:%M:%S")

    nonce: int = 0

    def hash\_block(self):

        sha = hashlib.sha256()

        record = str(self.record).encode()

        sha.update(record)

        creator\_id = str(self.creator\_id).encode()

        sha.update(creator\_id)

        timestamp = str(self.timestamp).encode()

        sha.update(timestamp)

        prev\_hash = str(self.prev\_hash).encode()

        sha.update(prev\_hash)

        nonce = str(self.nonce).encode()

        sha.update(nonce)

        return sha.hexdigest()

@dataclass

class PyChain:

    chain: List[Block]

    difficulty: int = 4

    def proof\_of\_work(self, block):

        calculated\_hash = block.hash\_block()

        num\_of\_zeros = "0" \* self.difficulty

        while not calculated\_hash.startswith(num\_of\_zeros):

            block.nonce += 1

            calculated\_hash = block.hash\_block()

        print("Wining Hash", calculated\_hash)

        return block

    def add\_block(self, candidate\_block):

        block = self.proof\_of\_work(candidate\_block)

        self.chain += [block]

    def is\_valid(self):

        block\_hash = self.chain[0].hash\_block()

        for block in self.chain[1:]:

            if block\_hash != block.prev\_hash:

                print("Blockchain is invalid!")

                return False

            block\_hash = block.hash\_block()

        print("Blockchain is Valid")

        return True

################################################################################

# Streamlit Code

# Adds the cache decorator for Streamlit

st.cache\_resource

def setup():

    print("Initializing Chain")

    return PyChain([Block("Genesis", 0)])

st.markdown("# PyChain")

st.markdown("## Store a Transaction Record in the PyChain")

pychain = setup()

################################################################################

# Step 3:

# Add Relevant User Inputs to the Streamlit Interface

# Code additional input areas for the user interface of your Streamlit

# application. Create these input areas to capture the sender, receiver, and

# amount for each transaction that you’ll store in the `Block` record.

# To do so, complete the following steps:

# 1. Delete the `input\_data` variable from the Streamlit interface.

# 2. Add an input area where you can get a value for `sender` from the user.

# 3. Add an input area where you can get a value for `receiver` from the user.

# 4. Add an input area where you can get a value for `amount` from the user.

# 5. As part of the Add Block button functionality, update `new\_block` so that `Block` consists of an attribute named `record`, which is set equal to a `Record` that contains the `sender`, `receiver`, and `amount` values. The updated `Block`should also include the attributes for `creator\_id` and `prev\_hash`.

# @TODO:

# Delete the `input\_data` variable from the Streamlit interface.

#input\_data = st.text\_input("Block Data")

# @TODO:

# Add an input area where you can get a value for `sender` from the user.

sender = st.text\_input("Input Sender Information")

# @TODO:

# Add an input area where you can get a value for `receiver` from the user.

receiver = st.text\_input("Input reciver Information")

# @TODO:

# Add an input area where you can get a value for `amount` from the user.

amount = st.number\_input("Transaction Amount")

if st.button("Add Block"):

    prev\_block = pychain.chain[-1]

    prev\_block\_hash = prev\_block.hash\_block()

    # @TODO

    # Update `new\_block` so that `Block` consists of an attribute named `record`

    # which is set equal to a `Record` that contains the `sender`, `receiver`,

    # and `amount` values

    new\_block = Block(

    # data=input\_data,

    record=Record (sender,receiver, amount),

    creator\_id=42,

    prev\_hash=prev\_block\_hash

    )

    pychain.add\_block(new\_block)

    st.balloons()

################################################################################

# Streamlit Code (continues)

st.markdown("## The PyChain Ledger")

pychain\_df = pd.DataFrame(pychain.chain).astype(str)

st.write(pychain\_df)

difficulty = st.sidebar.slider("Block Difficulty", 1, 5, 2)

pychain.difficulty = difficulty

st.sidebar.write("# Block Inspector")

selected\_block = st.sidebar.selectbox(

    "Which block would you like to see?", pychain.chain

)

st.sidebar.write(selected\_block)

if st.button("Validate Chain"):

    st.write(pychain.is\_valid())

################################################################################

# Step 4:

# Test the PyChain Ledger by Storing Records

# Test your complete `PyChain` ledger and user interface by running your

# Streamlit application and storing some mined blocks in your `PyChain` ledger.

# Then test the blockchain validation process by using your `PyChain` ledger.

# To do so, complete the following steps:

# 1. In the terminal, navigate to the project folder where you've coded the

#  Challenge.

# 2. In the terminal, run the Streamlit application by

# using `streamlit run pychain.py`.

# 3. Enter values for the sender, receiver, and amount, and then click the "Add

# Block" button. Do this several times to store several blocks in the ledger.

# 4. Verify the block contents and hashes in the Streamlit drop-down menu.

# Take a screenshot of the Streamlit application page, which should detail a

# blockchain that consists of multiple blocks. Include the screenshot in the

# `README.md` file for your Challenge repository.

# 5. Test the blockchain validation process by using the web interface.

# Take a screenshot of the Streamlit application page, which should indicate

# the validity of the blockchain. Include the screenshot in the `README.md`

# file for your Challenge repository.