Clase Blockchain

Vamos a desarrollar una blockchain que integre todos los capítulos anteriores.

Una blockchain es una cadena de bloques que se van anexando, y cada bloque se vincula con el bloque anterior. Los bloques incluyen transacciones (cambios de estado en la red). Las transacciones son hechas por los usuarios. La forma en la que se anexan los bloques a la red es por medio de un consenso (Proof of Work y Proof of stake)

Tenemos todo listo para hacer una Blockchain que integre todo lo que vimos con anterioridad.

Crearemos un archivo llamado blockchain.py en nuestro directorio de trabajo. En él, vamos a crear una nueva clase llamada Blockchain.

In [2]:
```
class Blockchain:
    pass
```

A diferencia de las clases anteriores, no vamos a necesitar mandar ninguna variable al instanciar un nuevo objeto Blockchain. Los ajustes los vamos a ir anexando por aparte.

Como atributos nuestra blockchain va a tener los siguientes atributos:

- chain: Lista que almacenara la cadena de bloques de la blockchain.
- tx_limit_per_block: Variable que define el límite de transacciones en espera antes de ser anexadas a la red.
- holding_tx: Lista que almacenara las transacciones en lista de espera.

Por parte del consenso Proof of Stake. Vamos a agregar atributos específicos.

- total_stacked: Número entero que define el dinero total en stack de toda la red.
- validators: Diccionario que almacena las direcciones de los validadores y su stack.
- last_block: Variable que almacena el próximo bloque a ser anexado a la red.

In [3]:
```
class Blockchain:
    def __init__(self):
        # Atributos
        self.protocol = None
        self.chain = []
        self.tx_limit_per_block = 1
        self.holding_tx = []
        # Atributos propios de PoS
        self.total_stacked = 0
        self.validators = {}
        self.last_block = None
```

Métodos

En esta sección vamos a explicar cada método y el porqué de su implementación.

Set Consensus

Necesitamos implementar un método que cambie por completo el sistema en el que nuestra blockchain funciona cambiando de consenso.

In [15]:
```
def set_consensus(self, protocol: str):
        if protocol not in ('PoW', 'PoS'):
            print('Seleccione un consenso de los dos disponibles.(PoW, PoS)')
            sys.exit()
        self.consensus = protocol
```

Genesis Block

Al momento de lanzar una blockchain a una red de prueba, lo primero es inicializarlo con un bloque génesis.

Cada bloque cuenta con su propio hash y el hash del bloque anterior, esto sirve para vincularse entre sí y darle inmutabilidad a la red; pero, ¿Qué pasa cuando hablamos del bloque 0? ¿Qué procede cuando el bloque 0 no tiene un bloque por detrás? ¿Cómo se vincula?

Hay que diseñar una función que nos ayude a resolver esta problemática.

In [11]:
```
def generate_genesis_block(self):
        """Inicializa el bloque genesis."""
        print('asdfas')
        if len(self.chain) == 0: # Comprueba que la blockchain este vacia.
            tx = Transaction(Account(0, "Genesis0"), 0, Account(0, "Genesis01"))
            block = Block('0', [tx], 0)
            self.mine(block)
            self.chain.append(block)
            block.list_of_transactions[0].change_status('CONFIRMADA')
        else:
            raise "Error: La Blockchain tiene que estar vacia."
```

Al comprobar que la blockchain esta vacía, puede procede a crear cuentas y transacciones que solo se usaran una vez para inicializar el bloque génesis.

¿Cómo recibe la blockchain las transacciones?

New Tx

Implementaremos un método que reciba una transacción hecha en local y la mande a todos los procesos que necesita para ser anexada a un bloque.

In [27]:
```
from account import Account

# Recibe los parametros de la transaccion.

def new_tx(self, _sender: Account, _value: int, _receiver: Account):
        """Recibe los parametros para instanciar un objeto de
        tipo Transaction; verifica que sea una transacción válida
        y lo añade a la holding list de la blockchain,
        para luego ser parte de un bloque. """
```

```
        # 0.5. Se debe verificar que quien manda la tx, tenga suficiente balance er
        if _value > _sender.balance:
            print()
            print('No tienes suficiente balance en tu cuenta.')
            return
        print('\n','Nueva transaccion detectada... Balance suficiente.')
        # 1. Instanciar un objeto transaccion.
        tx = Transaction(_sender, _value, _receiver)
        # 1.2 Al momento de instanciar el objeto, le restamos a la cuenta principal
        # el dinero que envio.
        _sender.balance -= _value
        print("Estado: {}".format(tx.status.name))
        # 2. Esta transaccion necesita ser firmada (confirmada).
        tx.sign_transaction()
        # 3. La firma pasa a ser verificada, con la finalidad de comprobar que sea
        # 3.1. Si es correcta signifa que puede agregarse a un bloque
        if tx.verify_signature() == True:
            if len(self.holding_tx) < self.tx_limit_per_block: # Revisa si aun cabe
                self.holding_tx.append(tx)
                print("Transaccion añadida a la espera.")
            if len(self.holding_tx) >= self.tx_limit_per_block: # Revisa si la list
                self.add_tx_to_block()
        else:
            return
```

¿Qué pasa cuando las tx en lista de espera estan listas para ser anexadas a un bloque?

Add tx to block

Nuestra blockchain va a tener un cierto número de transacciones límite, cuando el número de tx en lista de espera sea igual o mayor al límite, se llamara el método add_tx_to_block.

Este método se implementa dentro de la función new_tx.

Es el encargado de agrupar las transacciones en espera a un bloque, e implementar el bloque a la red. La preparación del bloque va a variar dependiendo del mecanismo de consenso actual de la blockchain.

Caso: Proof of Work

En el caso de ser Proof of Work, la metodología es la siguiente:

1. Instanciaremos un objeto Block, para ellos vamos a necesitar enviar:

- El hash del bloque anterior.
- La lista de transacciones que entraran al bloque.
- El número del bloque.

2. Enviar ese bloque a una función minado.

3. Minar el bloque para encontrar el nonce y un hash que cumpla con el target deseado.

4. Anexar el bloque a la cadena.

5. Vaciar la lista de transacciones en espera.
6. Agregar el block_number en las transacciones que ya forman parte de un bloque oficial en la cadena de bloques.
7. Cambiar el estatus de las transacciones dentro del último bloque y "enviar" el dinero a quien corresponda.

In [59]:
```
def add_tx_to_block(self):
        """Funcion que toma las transacciones en espera y procede a implementarlas
        para su posterior adision en la cadena de bloques."""
        print("### Creando nuevo bloque ###")
        print('### Bloque No. ', len(self.chain))
        _block_number = len(self.chain)
        # Consenso Proof of Work
        if self.consensus == 'PoW':
            print('En PoW')
            block = Block(previous_hash=self.chain[-1].hash,list_of_transactions=se
            self.mine(block) # Minar el bloque y hallar su nonce
            self.chain.append(block) # Anexar ese bloque a la red
            print("### Bloque creado. ###\n")
            self.holding_tx = [] # Vaciar la lista de transacciones pendiente
            for tx in block.list_of_transactions: # Bucle que asigna el block numbe
                tx.block = _block_number
            self.verify_latest_tx() # 6
            self.send_money_to_receivers()
```

Mine

Minar un bloque significa anexarlo a la red, pero primero hay que encontrar el nonce para vencer la dificultad. Implementaremos un método que reciba un bloque y lo pase por un proceso de "minado".

El proceso es idéntico a como lo vimos en el apartado de proof of work. Se obtiene el block header, se mina, se busca hash que venza al target, una vez encontrado se anexa el hash al bloque.

Este método se implementa dentro de la función add_tx_to_block.

In [5]:
```
def mine(self, block) -> None:
        """Funciona que mina el bloque.
        Funciona segun el protocolo de Proof of Stake. """
        print('Dentro de funcion minado...')
        # Primero obtenemos el string que contiene toda la informacion del bloque.
        block_header = json.dumps(block.get_block_header()).encode()
        block_hashed = SHA256.new(block_header)
        block_hash = block_hashed.hexdigest()
        # Proceso de minado
        if self.consensus == 'PoW':
        # 1. Dificultad
            difficulty_hash = 0x0000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
            difficult_decimal = 176684706477838432958329750074291851582748389687561
            # Sigue hasta que el hash sea menor o igual a la dificultad
            while int(block_hashed.hexdigest(), 16) >= difficulty_hash:
                block.nonce += 1 # Incremento del Nonce
                block_header = json.dumps(block.get_block_header()).encode() # Se c
```

```
                block_hashed = SHA256.new(block_header) # Pdd. Va a salir distinto
                print('Nonce Guess: ', block.nonce)
                print('Resultant Hash: ' + str(block_hashed.hexdigest()))
                print('Decimal value of hash: ' + str(int(block_hashed.hexdigest(),
                block_hash = block_hashed.hexdigest() # El bloque guarda el hash er
            print('Winner hash: ', block_hash)
            block.hash = block_hash
        if self.consensus == 'PoS':
            # En Proof of Stake vamos a saltar el proceso de minado.
            block.hash = block_hash
```

Verify latest tx

Función que cambia el status de una transacción a "confirmada".

Una transacción está confirmada cuando ya es parte de la cadena de bloques.

In [52]:
```
def verify_latest_tx(self):
        """Pone como confirmadas las transacciones que ya forman parte de la cadena
        Bloques original."""
        latest_block = self.chain[-1]
        block_transactions = latest_block.list_of_transactions
        for tx in block_transactions:
            tx.change_status('CONFIRMADA')
```

Send money to receivers

Función que envía el dinero, dependiendo del estatus de la transaccion.

- Si la transacción es confirmada, el dinero se envía al destinatario.
- Si la transacción es rechazada, el dinero se regresa al remitente.

En sí, en una blockchain el dinero no se envía. La blockchain cambia de "estado". Esta implementación funciona, pero no es exactamente la forma "verdadera".

In [6]:
```
def send_money_to_receivers(self):
        """Funcion que envía el dinero a los recipientes."""
        latest_block = self.chain[-1]
        block_transactions = latest_block.list_of_transactions
        for tx in block_transactions:
            # Condicional que cambia el estado de la transaccion a Declinada (Para
            # tx.change_status('DECLINADA')
            if tx.status.name == 'CONFIRMADA':
                tx.recipient.balance += tx.value
            if tx.status.name == 'DECLINADA':
                tx.sender.balance += tx.value
```

Caso: Proof of Stake

En un consenso Proof of Stake ya no hay mineros, hay validadores, que van interpretando dos roles, de Forjadores y Testigos, para anexar bloques a la red. Se desarrolla de la siguiente manera:

1. La red debe de tener validadores. Si no los tiene, se tienen que crear antes de hacer transacciones.

2. Llamar a la función select_the_forger() para seleccionar al nuevo forjador del bloque y a los no ganadores.
3. Los no ganadores del algoritmo de selección de forjador pasan a ser los nuevos testigos.
4. El forjador agrupa y verifica las transacciones que estén en espera.
5. El forjador crea un bloque (incluye las transacciones verificadas).
6. El forjador firma el bloque con su llave privada.
7. El forjador envía el bloque a la red.
8. Los testigos revisan que el trabajo que hizo el forjador sea válido. Si la mayor parte de los validadores están de acuerdo se llega a un acuerdo, y el bloque puede ser anexado a la red.
9. El forjador anexa el bloque a la red.
10. Se confirman las transacciones dentro del bloque y se "envía" el dinero a quien corresponda.

In [12]:
```
    def add_tx_to_block(self):
        """Funcion que toma las transacciones en espera y procede a implementarlas
        para su posterior adision en la cadena de bloques."""
        print("### Creando nuevo bloque ###")
        print('### Bloque No. ', len(self.chain))
        _block_number = len(self.chain)
        if self.consensus == 'PoS':
            print('En PoS:')
            forger, validators_with_out_forger = self.select_the_forger()
            # Los validadores no seleccionados pasan a ser objetos Attestors.
            attestors = Attestors(validators_with_out_forger)
            print('Testigos: ', [attestor.validator.account.nickname for attestor i
            print('Forjador: ', [forger.validator.account.nickname])
            # En proceso de verificar las tx...
            verified_tx = forger.verify_tx(self.holding_tx)
            forger.create_a_block(self.chain[-1].hash, verified_tx, _block_number)
            forger.sign_block()
            # el forjador manda el bloque a la red
            print('### Enviando el bloque a la red...')
            self.last_block = forger.broadcast_block()
            print('### Iniciando atestiguamiento del bloque...')
            if attestors.attest(self.last_block):
                self.chain.append(self.last_block)
                print("### Bloque creado. ###\n")
                self.holding_tx = []
                for tx in self.last_block.list_of_transactions:
                    tx.block = _block_number
                self.verify_latest_tx()
                self.send_money_to_receivers()
            else:
                print('No se pudo incluir el bloque.')
                return
```

1. La red debe de tener validadores. Si no los tiene, se tienen que crear antes de hacer transacciones.

Set Validators

Implementaremos un método que recibe una lista de cuentas interesadas en ser validadores y las instancíe como una clase Validator.

- Su balance es restado de sus cuentas, y guardado en la red.
- Se les otorga sus tokens correspondientes.
- Son agregados a los validores disponibles de la red.

In [138…
```
def set_validators(self, accounts):
        """Funcion que determina los validadores de la red."""
        for account in accounts:
            if account.balance >= 100: # Si tiene igual o mas de 100 en balance, pu
                new_validator = Validator(account) # Se crea un validador mandando
                self.total_stacked += new_validator.account.balance # Se anade el t
                new_validator.set_tokens(new_validator.account.balance) # Se anader
                self.validators.update({new_validator: new_validator.account.balanc
                new_validator.account.balance -= new_validator.account.balance # Se
```

2. Llamar a la función select_the_forger() para seleccionar al nuevo forjador del bloque y a los no ganadores.

Esta implementación ya la vimos en el capítulo de proof of stake. Recordemos que es una función qué implementa un algoritmo de tómbola, los participantes compran tickets, o en su caso Tokens, los Tokens entran a una lista, la lista se revuelve y se escoge al azar a un forjador, los no-ganadores pasan a ser los testigos. Se retorna el forjador y a los validadores no-ganadores.

In [139…
```
def select_the_forger(self):
        """Funcion que selecciona que validador va a ser el forjador del nuevo bloc
        # Dentro de esta funcion, se va a disenar un algoritmo que escoja al siguie
        #
        # Cada validador tiene un 'stack' dentro del total en la blockchain. Cada
        # moneda que ingresaron es un 'ticket' que les puede dar la posibilidad de
        # el siguiente forjador del bloque.
        # La cantidad de tickets va a depender de la cantidad de monedas que ingres

        # Este algoritmo escojera un ticket al azar. El dueno del ticker sera el nu
        # 1.- Lista que almacenara todos los tickets de la rifa
        pool = []
        # 2.- Bucle que recorrera a cada validador, y añadara sus tokens a la lista
        for validator in self.validators:
            pool += validator.tokens
        # 3.- Se revuelve la lista. (Como si fuera un sorteo.)
        print('Acumulando los tokens de los validadores en el servidor actual...')
        print(len(pool), '- tokens acumulados.')
        print('Revolviendo la lista...')
        pool = sample(pool, len(pool))
        print('Lista revuelta!')
        # Funcion que revisa los tokens de cada quien. Estaria chido ponerla como f
        contador = 0
        for validator in self.validators.keys():
            for token in pool:
                if token.owner.account.nickname == validator.account.nickname:
                    contador += 1
            print(validator.account.nickname, contador)
            contador = 0
        # Fin de la validacion
        # Aqui deberia de ir algo estilo, escojer el ticket ganador.
        ticket_winner = choice(pool)
        forger = Forger(ticket_winner.owner)
        print(f'El forjador del nuevo bloque sera... {forger.validator.account.nick
```

```
        # los validadores no ganadores del sorteo pasan a ser testigos.
        # los testigos estan encargados de revisar que el forjador haga lo correcto
        validators_with_out_forger = [validator for validator in self.validators]
        # se remueve el forjador, asi solo quedan los testigos
        validators_with_out_forger.remove(forger.validator)
        return forger, validators_with_out_forger
```

3. Los no ganadores del algoritmo de selección de forjador pasan a ser los nuevos testigos.

In [140…
```
def add_tx_to_block(self):
        if self.consensus == 'PoS':
            ...
            attestors = Attestors(validators_with_out_forger)
```

4. El forjador agrupa y verifica las transacciones que estén en espera.

Para este proceso el forjado utiliza el método verify_tx:

In [141…
```
def add_tx_to_block(self):
        if self.consensus == 'PoS':
            ...
            verified_tx = forger.verify_tx(self.holding_tx)
```

5. El forjador crea un bloque (incluye las transacciones verificadas).

In [142…
```
def add_tx_to_block(self):
        if self.consensus == 'PoS':
            ...
            forger.create_a_block(self.chain[-1].hash, verified_tx, _block_number)
```

6. El forjador firma el bloque con su llave privada.

In [143…
```
def add_tx_to_block(self):
        if self.consensus == 'PoS':
            ...
            forger.sign_block()
```

7. El forjador envía el bloque a la red.

In [144…
```
def add_tx_to_block(self):
        if self.consensus == 'PoS':
            ...
            self.last_block = forger.broadcast_block()
```

8. Los testigos revisan que el trabajo que hizo el forjador sea válido. Si la mayor parte de los validadores están de acuerdo se llega a un acuerdo, y el bloque puede ser anexado a la red.

In [145…
```
def add_tx_to_block(self):
        if self.consensus == 'PoS':
            ...
            if attestors.attest(self.last_block):
                print("Acuerdo concretado. El bloque puede anexarse a la red")

            else:
```

```
                print('No se pudo incluir el bloque.')
                return
```

9. El forjador anexa el bloque a la red.

In [146…

```
def add_tx_to_block(self):
        if self.consensus == 'PoS':
            ...
            if attestors.attest(self.last_block):
                print("Acuerdo concretado. El bloque puede anexarse a la red")
                self.chain.append(self.last_block)
                print("### Bloque creado. ###\n")
            else:
                print('No se pudo incluir el bloque.')
                return
```

10. Se confirman las transacciones dentro del bloque y se "envía" el dinero a quien corresponda.

In [147…

```
def add_tx_to_block(self):
        if self.consensus == 'PoS':
            ...
            if attestors.attest(self.last_block):
                print("Acuerdo concretado. El bloque puede anexarse a la red")
                self.chain.append(self.last_block)
                print("### Bloque creado. ###\n")

                # Se vacia la lista de tx pendientes
                self.holding_tx = []

                # se agrega el numero del bloque a la transacicon
                for tx in self.last_block.list_of_transactions:
                    tx.block = _block_number

                # se verifican las transacciones dentro del bloque
                self.verify_latest_tx()

                # el dinero se envia dependiendo del estado de las transacciones
                self.send_money_to_receivers()
            else:
                print('No se pudo incluir el bloque.')
                return
```

Esto seria todo de la clase Blockchain. Pasemos al siguiente capítulo para poner en práctica todo lo que hemos visto.