BlockchainSimulation

March 20, 2019

Clase Transacción

In [7]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Define los elementos que componen una transacción.
"""

class Transaccion(object):

    def __init__(self, emisor, receptor, cantidad):
        """
        Creación de una transaccióna
        :param emisor: Hash de quién envía
        :param receptor:  Hash de quién recibe
        :param cantidad: valor de la transacción
        """
        self._emisor = emisor
        self._receptor = receptor
        self._cantidad = cantidad

    def getEmisor(self):
        """
        Obtiene el valor actual del hash del emisor
        :return: Hash del emisor
        """
        return self._emisor;

    def getReceptor(self):
        """
        Obtiene el valor actual del hash del receptor
        :return: Hash del receptor
        """
        return self._receptor;

    def getCantidad(self):
        """
        Obtiene el valor de la transacción
```

```
        :return: Valor de la transacción
        """
        return self._cantidad;

    def __str__(self):
        """
        Genera un string con la información de la transacción
        :return: representación en String de una transacción
        """
        return "{} , {} , {}".format(self._emisor, self._receptor, self._cantidad)
```

Prueba Unitaria Clase Transacción

In [2]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from Transaccion import Transaccion
from Bloque import Bloque
from random import uniform
from random import random
import jsonpickle
import jsonpickle.backend
import jsonpickle.handlers
import pprint

"""
En esta clase se realizan pruebas de unidad
"""
class pruebasUnidadTransaccion(object):

    @staticmethod
    def jsonDefault(object):
        return object.__dict__

    def main():
       transacciones = [];
       #Se construye una conjunto aleatorio de transacciones
       for i in range(5):
           emisor = Bloque.generarBloqueHash(random())
           receptor = Bloque.generarBloqueHash(random())
           cantidad = uniform(2.5, 1000.0)
           transacciones.append(Transaccion(emisor, receptor,cantidad));

       print("CONJUNTO ALEATORIO DE TRANSACCIONES")
       pprint.pprint(jsonpickle.pickler.Pickler().flatten(transacciones))

    if __name__ == "__main__":
        main()
```

```
CONJUNTO ALEATORIO DE TRANSACCIONES
[{'_cantidad': 203.66456670821438,
  '_emisor': 'cfa82607adac99aaabae7cec8b9b4b719d871e987a00ebb7d9c5cc1ee93a6a76',
  '_receptor': '5024b513039bbf695b6f0a71fd8ed7bd865a4cc72d75d566a5fff597e34dab19',
  'py/object': 'Transaccion.Transaccion'},
 {'_cantidad': 834.4563379913702,
  '_emisor': '9750962eef171090b7e20898752b770d0ba241ef7e6ecb0e05359964cbec8a43',
  '_receptor': 'd66d52d6a2a2cefca7d98b6c957700fd41ff8b3a3d910698305fd16f7fe68d26',
  'py/object': 'Transaccion.Transaccion'},
 {'_cantidad': 161.6765922669576,
  '_emisor': 'ecda4062b102f1fad39bdc8bbbc081fc138f97bb62ed48e71c84b434d4c96e6c',
  '_receptor': '36ce88519a21b62d93d9f69d21f9013b076a35b50f89b277db1701a1bbfb8734',
  'py/object': 'Transaccion.Transaccion'},
 {'_cantidad': 798.8531049411085,
  '_emisor': '153441ec7145aa14388a54bfa84ff0dcf2422cb4d0e0f074704256312c406a7d',
  '_receptor': 'cf5252f833c3f5a9e173af1bb600b15a6cb80ba823196bbdf5bed2f735d5b2ae',
  'py/object': 'Transaccion.Transaccion'},
 {'_cantidad': 440.52259568593405,
  '_emisor': '9a27161fe22b29060eb2fe4afa3c83be1320432ae5b13d32d6037342ad189fe7',
  '_receptor': '28114126f9ac3f36fc36c70b5a0f4b4372406d0f11d74d78b701951d57f386fd',
  'py/object': 'Transaccion.Transaccion'}]
```

Clase Usuario

```
In [3]: #!/usr/bin/env python3
        # -*- coding: utf-8 -*-
        import time
        import hashlib
        from Transaccion import Transaccion
        """
        """


        class Usuario(object):

            def __init__(self, idUsuario):
                """
                Creación de un usuario en el sistema
                :param _idUsuario Identificador entero del usuario
                :param hashUsuario:  Código Hash del usuario
                :param saldo: Saldo del usuario
                :param marcaTiempo: Instante de tiempo de creación del usuario
                """
                self._idUsuario  = idUsuario;
                self._saldo = 0;
                self._hashUsuario = 0;
                self._marcaTiempo = time.time();
```

```
        """
        Usuario Génesis toma 100 de saldo
        """
        if self._idUsuario == 0:
            self._saldo = 100;


    def gethashUsuario(self):
        """
        Obtiene el valor actual del hash del usuario
        :return: Hash del bloque
        """
        return self._hashUsuario;

    def setHashUsuario(self):
        self._hashUsuario = self.generarUsuarioHash();


    def generarUsuarioHash(self):
        """
        Genera Hash la información del bloque
        :return: Hash construido a partir de la información del usuario
        """
        return hashlib.sha256(str(self).encode()).hexdigest()


    def __str__(self):
        """
        Genera un string con la información del usuario
        :return: representación en String del usuario
        """
        return "{}, {} , {}".format(self._idUsuario, self._hashUsuario, self._saldo)

    def enviar(self, valor, receptor):
        """
        :param receptor: hash del usuario receptor de la transferencia
        :param valor: Cantidad que se desea transferir
        :return: La nueva transacción que será enviada a validación
        """
        if valor < self._saldo:
            self._saldo -= valor;
            self.recibir(valor, receptor);
            return Transaccion(self._hashUsuario, receptor,valor);

    def recibir(self, valor, receptor):
        """
        :param receptor: hash del usuario receptor de la transferencia
        :param valor: Cantidad que se ha recibido
```

```
            """
            receptor._saldo += valor;
```

Prueba Unitaria Clase Usuario

```
In [4]: #!/usr/bin/env python3
        # -*- coding: utf-8 -*-
        from Usuario import Usuario


        """
        En esta clase se realizan pruebas de unidad
        """
        class pruebasUnidadUsuario(object):

            @staticmethod
            def jsonDefault(object):
                return object.__dict__

            def main():
               usuarios = {};
               for i in range(10):
                   temp = Usuario(i);
                   temp.setHashUsuario();
                   usuarios[temp.gethashUsuario] = temp;

               print("CONJUNTO DE USUARIOS")
               print("id\t hash \t saldo");
               for key in usuarios:
                   print(str(usuarios[key]));


            if __name__ == "__main__":
                main()
```

```
CONJUNTO DE USUARIOS
id         hash          saldo
0, ae1bc1b04561fac552570296dfe154094dc4df4b6d4970ca81ec9507046375b2 , 100
1, 9d5eab463378fe259f33d07590aa6a913a1d099d5a276e353a955cb2da4311f2 , 0
2, b4aec9b924aa7f7d83851c90a7f5142be0cec41c7c46cff2fb89bf0bbf055f67 , 0
3, 0890bee71d46a465a64625124eee9a424a66908e11ce283ed4afaf5a9551cef0 , 0
4, 1206cfe53ef4288f00f4c53f5377601c07bdc95f6a5249b95183cc5db54ad93a , 0
5, 5d4a62b58203901f4e7502ccbdd5a2e52a234bb44545549fd5c87edef9202363 , 0
6, 11446965a28709ff7577c812099db7093ae7a31f8aa8ebe2a81d15ee876b0f16 , 0
7, 44b70f5a64febf0f1c47390efaa1d824a5503e3831d37b28b42c63f25813cf17 , 0
8, 06aeb9cf521c8e7114647117303ebf837f50bd39b38104ce5e3371bf54d1db61 , 0
9, 371eb3f89a125644770e3395faa25a50d0f7bf6f9fb15b271649b53640ed1a3b , 0
```

Clase Bloque

In [5]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import time
import hashlib
import copy
import json
"""
Define la estructura del bloque.
"""

class Bloque(object):

    def __init__(self, indice, minado, hashBloque, transacciones, marcaTiempo=None):
        """
        Inicialización del bloque
        :param indice: Valor entero que representa la posición de cada bloque en la ca
        :param minado: Número entero usado en el proceso de minería
        :param hashBloque:  hash del bloque actual en la cadena
        :param transacciones: Conjunto de transacciones del bloque
        :param marcaTiempo: Instante de tiempo de creación del nuevo bloque
        """
        self._indice = indice
        self._minado = minado
        self._hashBloque = hashBloque
        self._transacciones = copy.copy(transacciones)
        self._marcaTiempo = marcaTiempo or time.time()

    def gethashBloque(self):
        """
        Obtiene el valor actual del hash del bloque
        :return: Hash del bloque
        """
        return self._hashBloque;

    def setHashBloque(self):
        self._hashBloque = Bloque.generarBloqueHash(self)

    @staticmethod
    def generarBloqueHash(self):
        """
        Genera Hash la información del bloque
        :return: Hash construido a partir de la información del bloque
        """
        return hashlib.sha256(str(self).encode()).hexdigest()
```

```
        def __str__(self):
            """
            Genera un string con la información del bloque
            :return: representación en String del bloque
            """
            return "{} , {} , {}  , {}".format(self._indice, self._minado, self._hashBloque

        def __repr__(self):
                return json.JSONEncoder().encode(self)
```

Prueba Unitaria Clase Bloque

```
In [6]: #!/usr/bin/env python3
        # -*- coding: utf-8 -*-
        from Transaccion import Transaccion
        from Bloque import Bloque
        from random import randrange
        from random import uniform
        from random import random
        import jsonpickle
        import jsonpickle.backend
        import jsonpickle.handlers
        import pprint

        """
        En esta clase se realizan pruebas de unidad
        """
        class pruebasUnidadBloque(object):

            def main():
               #Cada bloque de prueba se ha creado con un conjunto aleatorio transacciones
               transacciones = [];
               #Se construye una conjunto aleatorio de transacciones para bloque genésis
               emisor = Bloque.generarBloqueHash(random())
               receptor = Bloque.generarBloqueHash(random())
               cantidad = uniform(2.5, 1000.0)
               transacciones.append(Transaccion(emisor, receptor,cantidad));
               #0 para indicar que el bloque no ha sido minado
               genesis = Bloque(0,0,Bloque.generarBloqueHash(0),transacciones);
               cadena = [genesis];

                #Se construye una pseudocadena con 5 bloques
               for i in range(1,5):
                      transacciones = [];
                      #Se construye una conjunto aleatorio de transacciones
                      for t in range(1+randrange(3)):
                          emisor = Bloque.generarBloqueHash(random())
                          receptor = Bloque.generarBloqueHash(random())
```

```
                        cantidad = uniform(2.5, 1000.0)
                        transacciones.append(Transaccion(emisor, receptor,cantidad));

                    cadena.append(Bloque(i,0,Bloque.generarBloqueHash(cadena[i-1].gethashBlo

            pprint.pprint(jsonpickle.pickler.Pickler().flatten(cadena))

        if __name__ == "__main__":
            main()
```

```
[{'_hashBloque': '5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9',
  '_indice': 0,
  '_marcaTiempo': 1529247369.137099,
  '_minado': 0,
  '_transacciones': [{'_cantidad': 877.3810119704357,
                      '_emisor': 'a9df00a99c4be24e257cbff30f230f5190b2d5ab36c4277267cfeae5631a
                      '_receptor': '21d5eca5521a35d601c86987fbc3ad932a29278b894b2015580902e894
                      'py/object': 'Transaccion.Transaccion'}],
  'py/object': 'Bloque.Bloque'},
 {'_hashBloque': '5122a1d1bc9d87662dcf5fb870adf8c55faf2ce12fad1bacac2fe7df88172466',
  '_indice': 1,
  '_marcaTiempo': 1529247369.1371431,
  '_minado': 0,
  '_transacciones': [{'_cantidad': 883.2276154566855,
                      '_emisor': '0ed246acb2dbb5b13218ff358c86629997caa3d9b420f41a58dc8a19b545
                      '_receptor': 'eaf44a27bb108fbb343214593237a7d556c4519b03ad8f270f4fe77062
                      'py/object': 'Transaccion.Transaccion'},
                     {'_cantidad': 276.6887479260855,
                      '_emisor': 'e0fc82ae30827fc813242d95aeecd56fbcbdcc11275ff771c85f6df1f62c
                      '_receptor': 'd9c8eb2e75add7b6aa45cf56f753c0a8b02ed13e62f2cbe0650e6c00e1
                      'py/object': 'Transaccion.Transaccion'}],
  'py/object': 'Bloque.Bloque'},
 {'_hashBloque': '3c857ab8e3936f0f125a97a86262fd3d1cd906f4b7acf1df385f1756f7f692b2',
  '_indice': 2,
  '_marcaTiempo': 1529247369.1371617,
  '_minado': 0,
  '_transacciones': [{'_cantidad': 197.136994685999,
                      '_emisor': 'fdad280e38116f37382db3633df35619589d9176cd80b05b5c5ac29a4fce
                      '_receptor': '259885aa40fad971e3fe0a945f15d229f6a9f9b10858ab3edbeda090ea
                      'py/object': 'Transaccion.Transaccion'}],
  'py/object': 'Bloque.Bloque'},
 {'_hashBloque': '3806743a62ad600a453105090db2884c747600ad27576f4e6ba18f2f56cc445b',
  '_indice': 3,
  '_marcaTiempo': 1529247369.1371787,
  '_minado': 0,
  '_transacciones': [{'_cantidad': 14.972984892336266,
                      '_emisor': 'd939634d05ed10d16ec0d1a429bf22dfb4f3dd3a9cebf0f7561949623ba5
                      '_receptor': '489d4d3461255a694e63ca6edfd22be63156b786582073ca09108632f7
```

```
                    'py/object': 'Transaccion.Transaccion'}],
  'py/object': 'Bloque.Bloque'},
 {'_hashBloque': '19870586ed9e6d23573bd38b361f090ec70a0260fbf5b6e9d667f239912ccf94',
  '_indice': 4,
  '_marcaTiempo': 1529247369.1372035,
  '_minado': 0,
  '_transacciones': [{'_cantidad': 64.61594012977609,
                      '_emisor': '55ee9138f8528cb62a3742b8a3c107fe6e83c86a2504f1fd605b56e462b4
                      '_receptor': '4e2620109ea5f93d34900c5694fc0010d8b07f986a179c8582aacdf969
                      'py/object': 'Transaccion.Transaccion'},
                     {'_cantidad': 770.3643679591898,
                      '_emisor': 'fefcb5c9349bc1c439102a5850fbf3deda3772d4a7e0faf2ee9f0e320c7a
                      '_receptor': 'df22b53cfb72b48b4a9a4c1a4b339abe9eb9994ff77c1593a76d6d5587
                      'py/object': 'Transaccion.Transaccion'}],
  'py/object': 'Bloque.Bloque'}]
```

Clase Nodo Minador

In [8]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Define la estructura de un nodo responsable de minar la cadena de bloques
"""
import time
import copy

class NodoMinador(object):

    def __init__(self, direccion, puerto, descripcion, usuariosMineros):
        """
        Creación de un nodo para la minería de blockchain
        :param _direcccción: Dirección ip del nodo minero
        :param _puerto:  puerto TCP/UDP donde se publica el servicio
        :param _descripcion: información adicional del nodo minador
        :param _fechaCreacion: fecha de adición del nodo a la red
        :param _cadenaBloques: Recibe una copia de la cadena de bloques
        :param _hashRateNodo : Capacidad de resolución de hash por segundo del nodo
        :param _usuariosMineros: Conjunto de usuarios participantes en el nodo minero
        """
        self._direccion = direccion;
        self._puerto = puerto;
        self._descripcion = descripcion;
        self._utilidad = 0;
        self._cantidadBloquesMinados = 0;
        self._fechaCreacion = time.time();
        self._hashRateNodo = 100;
        self._cadenaBloques = [];
```

```
            self._usuariosMineros = copy.copy(usuariosMineros);


        def actualizarCadena(self, nuevaCadena):
            """
            El proceso de minería requiere que el nodo actualice la blockchain
            :param: Cadena de bloques actualizada
            """
            self._cadenaBloques = copy.copy(nuevaCadena)

        def infoCadena(self):
            """
            Genera un string con la información básica de la cadena de bloques
            :return: una cadena con el reporte del total de nodos de la cadena
            """
            return "La cadena tiene {} bloques".format(str(len(self._cadenaBloques)))

        def __str__(self):
            """
            Genera un string con la información de un nodo de minería
            :return: representación en String del nodo
            """
            return "{} , {} , {} , {} , {}".format(self._descripcion , str(self._direccion)
```

Prueba Unitaria Clase Nodo Minador

```
In [9]: #!/usr/bin/env python3
        # -*- coding: utf-8 -*-
        from CadenaBloques import CadenaBloques
        from NodoMinador import NodoMinador
        from Transaccion import Transaccion
        from Usuario import Usuario
        from Bloque import Bloque
        from random import randrange
        from random import uniform
        from random import random
        import copy

        """
        En esta clase se realizan pruebas de unidad
        """
        class pruebasUnidadNodoMineria(object):
            def main():
                transacciones = [];
                blockchain = CadenaBloques()
                #Se construye una transacción aleatoria para bloque genésis
                emisor = Bloque.generarBloqueHash(random())
```

```
    receptor = Bloque.generarBloqueHash(random())
    cantidad = uniform(2.5, 1000.0)
    transacciones.append(Transaccion(emisor, receptor,cantidad));
    blockchain.bloqueGenesis(transacciones)

    totalUsuarios = 25;
    totalMineros = 5;

    usuarios = {};
    for i in range(totalUsuarios):
        temp = Usuario(i);
        temp.setHashUsuario();
        usuarios[temp.gethashUsuario] = temp;

    #Se construye una conjunto aleatorio de nodos mineros
    usuariosNodo = [];
    usuariosMineros = {};
    contador = totalUsuarios//totalMineros - 1;
    aux = 0;
    for key in usuarios:
        usuariosMineros[key] = usuarios[key];
        if(aux < contador):
           aux += 1;
        else:
            usuariosNodo.append(copy.copy(usuariosMineros));
            aux = 0;
            usuariosMineros = {};

    mineros = [];
    for i in range(len(usuariosNodo)):
        address = "127.0.0." + str(1+randrange(253))
        puerto = 1025+randrange(60000)
        descripcion = "nodo minero número " + str(i)
        nodo = NodoMinador(address, puerto,descripcion,usuariosNodo[i])
        nodo.actualizarCadena(blockchain)
        mineros.append(nodo);

    print("CONJUNTO ALEATORIO DE NODOS MINEROS")
    #pprint.pprint(jsonpickle.pickler.Pickler().flatten(mineros))
    for i in mineros:
        print(i);
        print("Usuarios Mineros en el Nodo");
        for key in i._usuariosMineros:
            print(i._usuariosMineros[key]);
        print("============================================================")


if __name__ == "__main__":
```

```
                main()

CONJUNTO ALEATORIO DE NODOS MINEROS
nodo minero número 0 , 127.0.0.143:45186 , 0 , 0 , 5
Usuarios Mineros en el Nodo
0, ae1bc1b04561fac552570296dfe154094dc4df4b6d4970ca81ec9507046375b2 , 100
1, 9d5eab463378fe259f33d07590aa6a913a1d099d5a276e353a955cb2da4311f2 , 0
2, b4aec9b924aa7f7d83851c90a7f5142be0cec41c7c46cff2fb89bf0bbf055f67 , 0
3, 0890bee71d46a465a64625124eee9a424a66908e11ce283ed4afaf5a9551cef0 , 0
4, 1206cfe53ef4288f00f4c53f5377601c07bdc95f6a5249b95183cc5db54ad93a , 0
============================================================
nodo minero número 1 , 127.0.0.136:31969 , 0 , 0 , 5
Usuarios Mineros en el Nodo
5, 5d4a62b58203901f4e7502ccbdd5a2e52a234bb44545549fd5c87edef9202363 , 0
6, 11446965a28709ff7577c812099db7093ae7a31f8aa8ebe2a81d15ee876b0f16 , 0
7, 44b70f5a64febf0f1c47390efaa1d824a5503e3831d37b28b42c63f25813cf17 , 0
8, 06aeb9cf521c8e7114647117303ebf837f50bd39b38104ce5e3371bf54d1db61 , 0
9, 371eb3f89a125644770e3395faa25a50d0f7bf6f9fb15b271649b53640ed1a3b , 0
============================================================
nodo minero número 2 , 127.0.0.81:20701 , 0 , 0 , 5
Usuarios Mineros en el Nodo
10, d3e0b66ec1e56b92a5c0b1bd7c57540697da963987253413c40b77636707f13c , 0
11, b29492fe4bf566a54a5ac57bfb49cd6441b5cc3dbb19b36804da13b6a7495032 , 0
12, e8a26469e9a9879b4ef23fb12236e77a66b7e1a55b060694537c39fdca4fec8c , 0
13, 526042ccba454c2762af9fefd0d418254372e3d3c9d8605dfa6217d1c5843a61 , 0
14, 7f74614d41796a901e5dd4fc962da90a80b2e0a7a9f3180d968697845d882e55 , 0
============================================================
nodo minero número 3 , 127.0.0.163:56998 , 0 , 0 , 5
Usuarios Mineros en el Nodo
15, c1f330b2fa35be43c2d2e695cc0d73ba2c074247ef4e10a82e4a4eca63307c80 , 0
16, da6a1600010abc6c5a6c9aea97886cf0559fe1ab067110a0dabb3e6c449c3835 , 0
17, 33d772cef882b98fe5495ddc7b1e7b369fb6301372761b36542aa7f2683a0788 , 0
18, 5fafa6531ad8489b778a631289e5d41cc2a6dee01b2d17dab578b9ecc47abdac , 0
19, ce12b6eece7dc590ac7af2b77258ba8c262d5d0b1f89b8aff89914e6eb3a86b9 , 0
============================================================
nodo minero número 4 , 127.0.0.232:41227 , 0 , 0 , 5
Usuarios Mineros en el Nodo
20, b252c76b28f15c2fce8bfb7b0153ce87336a0789e16c40fed7faa04df2fc864f , 0
21, 839604719f7a03403bda1bea57e369f0ee226a4aa8a092aee337252b330e7ab8 , 0
22, 18fa0626ea0690666f53c910244389a32c4fa7a13453987b094ddd813ec1904d , 0
23, 192990181fd61e86ffa1e28ec8741a76c1640e816bd75b00e8e7e4499b12c1af , 0
24, 1d918ed33bf67966f2ab8ea55bb3707f2ead718e01b6881c5ec03882587760e1 , 0
============================================================
```

Clase Cadena de Bloques

```
In [10]: #!/usr/bin/env python3
         # -*- coding: utf-8 -*-
```

```
"""
En esta clase almacena la cadena de bloques
"""

from Bloque import Bloque
from PoolMinado import PoolMinado
import json
import copy

class CadenaBloques(object):

    def __init__(self):
        """
        Inicialización de la blockchain
        :param _cadena: Estructura de datos tipo array para almacenar la cadena de bl
        """
        self._cadena = []


    def getCadenaSerializada(self):
        """
        Serializa la cadena de bloques en un documento JSON
        :return conjunto de bloques en formato JSON
        """
        salida = ""
        for i in range(len(self._cadena)):
            salida = salida + str(self._cadena[i]) + "\nFin Bloque " + str(i) + str("
        return salida

    def bloqueGenesis(self,trs):
        """
        Crea el primer bloque en la cadena. Es el bloque seminal.
        """
        self.nuevoBloque(2, Bloque.generarBloqueHash(0), trs)

    def nuevoBloque(self, minado, hashBloque,trans):
        """
        Construye e inserta el siguiente bloque de la cadena
        :param minado Método de minado del bloque
        :param hashBloque Hash generado para el bloque
        :return Bloque nuevo bloque generado
        """
        bloque = Bloque(
            indice = len(self._cadena),
            minado = minado,
            hashBloque = hashBloque,
            transacciones=copy.copy(trans)
        )
```

```
        self._cadena.append(bloque)
        return bloque


    def obtenerUltimoBloque(self):
        """
        Devuelve una referencia al último nodo de la blockchain
        @return último nodo
        """
        return self._cadena[-1]

    @staticmethod
    def bloqueEsValido(nuevoBloque, ultimoBloque):
        """
        Verifica bajo cuatro reglas básicas la validez de un Bloque
        Regla 1: Secuencia ordenada de bloques por valor del índice
        Regla 2: Secuencia ordenada de bloques por fecha de creación
        Regla 3: Hash diferente entre el último bloque de la cadena y el nuevo bloque
        Regla 4: Bloque correctamente minado
        @return True si el bloque es válido, False en otro caso
        """
        if ultimoBloque._indice + 1 != nuevoBloque._indice:
            return False
        elif nuevoBloque._marcaTiempo <= ultimoBloque._marcaTiempo:
            return False
        elif ultimoBloque.gethashBloque != nuevoBloque.gethashBloque:
            return False
        elif not CadenaBloques.esPrimo(nuevoBloque.minado + ultimoBloque.minado):
            return False
        return True



    def validarBlockchain(self):
        """
        Verifica la integridad de la cadena de bloques.
        para este procedimiento se valida que el índice del siguiente bloque sea
        menor en una unidad del bloque anterior y que el hash de cada bloque sea
        correcto según la política de minado establecida, números primos en este caso
        @return True si la cadena esta bien formada, False en otro caso
        """
        bloqueAnterior = self._cadena[0]
        for i in range (1,len(self._cadena)):
            bloqueActual = self._cadena[i]
            if (bloqueAnterior._indice + 1) - bloqueActual._indice != 0:
                print("indice malo")
                return False
            if not PoolMinado.esPrimo(bloqueAnterior._minado + bloqueActual._minado):
```

```
                    return False
                bloqueAnterior = copy.copy(bloqueActual)

            return True



    def __repr__(self):
        return json.JSONEncoder().encode(self._cadena)
```

Prueba Unitaria Clase Cadena de Bloques

```
In [11]: #!/usr/bin/env python3
         # -*- coding: utf-8 -*-
         from CadenaBloques import CadenaBloques
         from NodoMinador import NodoMinador
         from Transaccion import Transaccion
         from Bloque import Bloque
         from PoolMinado import PoolMinado
         from Usuario import Usuario
         from random import randrange
         from random import uniform
         from random import random
         import jsonpickle
         import jsonpickle.backend
         import jsonpickle.handlers
         import pprint

         """
         En esta clase se realizan pruebas de unidad
         """
         class pruebasUnidadCadena(object):
             def main():
                pool = PoolMinado()
                transacciones = [];
                blockchain = CadenaBloques()
                #Se construye una transacción aleatoria para bloque genésis
                emisor = Bloque.generarBloqueHash(random())
                receptor = Bloque.generarBloqueHash(random())
                cantidad = uniform(2.5, 1000.0)
                transacciones.append(Transaccion(emisor, receptor,cantidad));
                blockchain.bloqueGenesis(transacciones)

                print("Agregando nodos mineros para la blockchain ...");
                totalUsuarios = 25;
                totalMineros = 5;

                usuarios = {};
```

```
        for i in range(totalUsuarios):
            temp = Usuario(i);
            temp.setHashUsuario();
            usuarios[temp.gethashUsuario] = temp;

        #Se construye una conjunto aleatorio de nodos mineros
        usuariosNodo = [];
        usuariosMineros = {};
        contador = totalUsuarios//totalMineros - 1;
        aux = 0;
        for key in usuarios:
            usuariosMineros[key] = usuarios[key];
            if(aux < contador):
               aux += 1;
            else:
                usuariosNodo.append(copy.copy(usuariosMineros));
                aux = 0;
                usuariosMineros = {};

        mineros = [];
        for i in range(len(usuariosNodo)):
            address = "127.0.0." + str(1+randrange(253))
            puerto = 1025+randrange(60000)
            descripcion = "nodo minero número " + str(i)
            nodo = NodoMinador(address, puerto,descripcion,usuariosNodo[i])
            nodo.actualizarCadena(blockchain)
            mineros.append(nodo);

        #Se construye una cadena de bloques con 10 bloques
        for i in range(1,10):
               transacciones = [];
               #Se construye una conjunto aleatorio de transacciones
               for tr in range(1+randrange(3)):
                   emisor = Bloque.generarBloqueHash(random())
                   receptor = Bloque.generarBloqueHash(random())
                   cantidad = uniform(2.5, 1000.0)
                   transacciones.append(Transaccion(emisor, receptor,cantidad))

               pool.torneoPorNuevoBloque(transacciones)
               print("Iteracion " + str(i) +" La cadena es válida? " + str(blockchain.v

        pprint.pprint(jsonpickle.pickler.Pickler().flatten(blockchain))


    if __name__ == "__main__":
        main()
```

```
Agregando nodos mineros para la blockchain ...
```

```
-
Iteracion 1 La cadena es válida? True ahora tiene 1 bloques
-
Iteracion 2 La cadena es válida? True ahora tiene 1 bloques
-
Iteracion 3 La cadena es válida? True ahora tiene 1 bloques
-
Iteracion 4 La cadena es válida? True ahora tiene 1 bloques
-
Iteracion 5 La cadena es válida? True ahora tiene 1 bloques
-
Iteracion 6 La cadena es válida? True ahora tiene 1 bloques
-
Iteracion 7 La cadena es válida? True ahora tiene 1 bloques
-
Iteracion 8 La cadena es válida? True ahora tiene 1 bloques
-
Iteracion 9 La cadena es válida? True ahora tiene 1 bloques
{'_cadena': [{'_hashBloque': '5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9
              '_indice': 0,
              '_marcaTiempo': 1529247715.5939398,
              '_minado': 2,
              '_transacciones': [{'_cantidad': 905.2468928548867,
                                  '_emisor': '888f882083818ee6cc2298e1d918d041fbb4e0d51e0ff3d2
                                  '_receptor': '930cf99d263c3695e622b1173b56c487b174ce9a1f5b33
                                  'py/object': 'Transaccion.Transaccion'}],
              'py/object': 'Bloque.Bloque'}],
 'py/object': 'CadenaBloques.CadenaBloques'}
```

Clase Pool de Minado

```
In [12]: #!/usr/bin/env python3
         # -*- coding: utf-8 -*-
         """
         Define la organización para los nodos que van a minar la cadena.
         """
         from random import random
         from multiprocessing.pool import ThreadPool
         import numpy as np
         from random import randint


         class PoolMinado(object):

             def __init__(self):
                 """
                 Inicialización del pool de minería
```

```
        :param _nodos: conjunto de nodos mineros
        :param _hasRate Simula la capacidad de computo del pool de mineria
        :param _recompensaPorBloque cantidad que se recompensa a los nodos que resuel
        :param _dificultad  valor usado para simular el incremento de dificultad del
        """
        self._nodos =  [];
        self._hashRate = 0;
        self._recompensaPorBloque = 10;
        self._dificultad = 1000;

    def adicionarMinero(self, minador):
        """
        Agrega  un nodo minero al conjunto de nodos minadores de la blockchain. Si el
        el hashrate del pool ylo iguala o supera el 60% de la dificultad, esta última
        20%
        @param minador: Nuevo nodo minero
        @return True si el nodo de mineria se puede adicionar, False en otro caso
        """
        self._nodos.append(minador)
        self._hashRate += minador._hashRateNodo
        if self._dificultad * 0.6 < self._hashRate:
            self._dificultad *= 1.2;

        return True

    @staticmethod
    def pruebaDeTrabajo(indiceUltimoMinado):
        """
        Algoritmo de minado por prueba de trabajo.
        En esta versión se implementa una prueba basada en números primos, en la cual
        valor de minado del bloque actual y del último bloque de la cadena debe ser u
        @param indiceUltimoMinado, Valor generado por la minería para el último bloqu
        @return minadoNuevo Valor de minado obtenido para el nuevo bloque
        """
        minadoNuevo = indiceUltimoMinado + 1
        while not PoolMinado.esPrimo(minadoNuevo + indiceUltimoMinado):
            minadoNuevo += 1

        return minadoNuevo

    @staticmethod
    def esPrimo(numero):
        """
        Determina si un número es primo
        @param numero, Número que necesitamos determinar si es primo
        @return True si numero es primo, False en otro caso
        """
        if numero < 2:
```

```
            return False
        elif numero == 2:
            return True
        else:
            for x in range(2,numero):
                if(numero % x==0):
                    return False
        return True

    def minarBloque(self,nodoMinero, numeroNodo, output):
        """
        Algoritmo que realiza el minado de un bloque para un nodo en especifico.
        @param nodoMinero, Nodo minero que realizará la prueba de trabajo
        @param numeroNodo, identificador entero del nodo minero en el pool de minería
        @param output, Parámetro de salida donde se conserva la información de
        la prueba de trabajo ejecutada por el iésimo nodo minero.
        @return minadoNuevo Valor de minado obtenido para el nuevo bloque
        """
        trabajo = 0
        for i in range(self._dificultad - nodoMinero._hashRateNodo + randint(1,10000)):
            trabajo += 1
        ultimo = nodoMinero._cadenaBloques.obtenerUltimoBloque()
        nuevoValorMinado = PoolMinado.pruebaDeTrabajo(ultimo._minado)
        #print("[" + nodoMinero._descripcion + "] ha trabajado " + str(trabajo) + " n
        output[numeroNodo][0] = numeroNodo;
        output[numeroNodo][1] = nuevoValorMinado;
        output[numeroNodo][2] = trabajo;


    def torneoPorNuevoBloque(self,transacciones):
        """
        Realiza un torneo para la adjudicación del nuevo bloque a un nodo minero.
        El nodo ganador será el que realice mayor trabajo.
        @param transacciones, Bloque de transacciones que tendrá el nuevo bloque de l
        """
        try:
            #print("Total nodos " + str(len(self._nodos)))
            numOfThreads = int(len(self._nodos))
            pool = ThreadPool(numOfThreads)
            output = [];
            for x in range(numOfThreads):
                output.append([0]*3)

            for i in range(numOfThreads):
                pool.apply_async(self.minarBloque(self._nodos[i],i,output))
            pool.close()
            pool.join()
            utilidadObtenida = np.max(output);
```

```
                ganador = np.argmax(output)//(numOfThreads-1)+randint(0,1);
                #print("utilidad obtenida " + str(utilidadObtenida))
                self._nodos[ganador]._cantidadBloquesMinados += 1;
                self._nodos[ganador]._utilidad += utilidadObtenida;
                self.minadoBloque(self._nodos[ganador],transacciones);
                nueva = self._nodos[ganador]._cadenaBloques;
                self.sincronizarCadena(nueva);

                totalMinadores = len(self._nodos[ganador]._usuariosMineros);
                for key in self._nodos[ganador]._usuariosMineros:
                    self._nodos[ganador]._usuariosMineros[key]._saldo += utilidadObtenida/
                    #print(self._nodos[ganador]._usuariosMineros[key]._saldo);
                #print("=======================================================")

            except:
                #print ("Error: un hilo ha fallado. ",sys.exc_info())
                print("-");
    @staticmethod
    def minadoBloque(nodoMinero, transacciones):
        """
        Realiza el proceso de minado de un nuevo bloque en la cadena
        @param nodoMinero que realiza el intento de minar el bloque
        @param transacciones: Conjunto de Transacciones del nuevo bloque
        """
        ultimo = nodoMinero._cadenaBloques.obtenerUltimoBloque();
        nodoMinero._cadenaBloques.nuevoBloque(PoolMinado.pruebaDeTrabajo(ultimo._minad

    def sincronizarCadena(self,nuevaCadena):
        """
        Realiza una sincronización de la cadena actualizada a todos los mineros
        @param nuevaCadena, Cadena con el nuevo bloque
        """
        for i in range(len(self._nodos)):
            self._nodos[i].actualizarCadena(nuevaCadena);

    def __str__(self):
        """
        Genera un string con la información del pool de minería
        :return: representación en String del nodo
        """
        a = "";
        for i in range(len(self._nodos)):
            a += str(self._nodos[i]) + "\n";
        return a;
```

Prueba Unitaria Clase Pool de Minado

```
In [13]: #!/usr/bin/env python3
         # -*- coding: utf-8 -*-
```

```
from CadenaBloques import CadenaBloques
from Usuario import Usuario
from NodoMinador import NodoMinador
from Transaccion import Transaccion
from Bloque import Bloque
from PoolMinado import PoolMinado
from random import randrange
from random import uniform
from random import random
from threading import Thread

"""
En esta clase se realizan pruebas de unidad
"""
class pruebasUnidadPoolMinado(Thread):
    def main():
       pool = PoolMinado()
       transacciones = [];
       blockchain = CadenaBloques()
       #Se construye una transacción aleatoria para bloque genésis
       emisor = Bloque.generarBloqueHash(random())
       receptor = Bloque.generarBloqueHash(random())
       cantidad = uniform(2.5, 1000.0)
       transacciones.append(Transaccion(emisor, receptor,cantidad));
       blockchain.bloqueGenesis(transacciones)
       print("La cadena tiene un " + str(len(blockchain._cadena)) + " Bloque, el genés
       print("Agregando nodos mineros para la blockchain ...");

       maxUsuarios = 25;
       maxMineros = 5;
       maxTransaccionesPorBloque = 10;
       totalBloquesCaena = 100;

       usuarios = {};
       for i in range(maxUsuarios):
           temp = Usuario(i);
           temp.setHashUsuario();
           usuarios[temp.gethashUsuario] = temp;

       #Se construye una conjunto aleatorio de nodos mineros
       usuariosNodo = [];
       usuariosMineros = {};
       contador = maxUsuarios//maxMineros - 1;
       aux = 0;
       for key in usuarios:
           usuariosMineros[key] = usuarios[key];
           if(aux < contador):
              aux += 1;
```

```
                    else:
                        usuariosNodo.append(usuariosMineros);
                        aux = 0;
                        usuariosMineros = {};

                for i in range(len(usuariosNodo)):
                    address = "127.0.0." + str(1+randrange(253))
                    puerto = 1025+randrange(60000)
                    descripcion = "nodo minero " + str(i)
                    minerito = NodoMinador(address, puerto,descripcion,usuariosNodo[i]);
                    minerito.actualizarCadena(blockchain);
                    pool.adicionarMinero(minerito);


                print("agregados " + str(len(usuariosNodo)) + " mineros ...");
                print("Iniciando la simulación de crecimiento de la blockchain");
                #Se construye una cadena de bloques con 10 bloques
                for i in range(1,totalBloquesCaena):
                       transacciones = [];
                       #Se construye una conjunto aleatorio de transacciones
                       for tr in range(1+randrange(maxTransaccionesPorBloque)):
                           emisor = Bloque.generarBloqueHash(random())
                           receptor = Bloque.generarBloqueHash(random())
                           cantidad = uniform(2.5, 1000.0)
                           transacciones.append(Transaccion(emisor, receptor,cantidad))
                       #print("Bloque de transacciones: " + str(i) + " , Agregadas " + str(tr+
                       pool.torneoPorNuevoBloque(transacciones)
                       #print("La cadena tiene ahora " + str(len(blockchain._cadena)) + " Bloq
                print("Fin de la simulación");
                print("{Descripcion} , {Dirección} , {Puerto} , {Utilidad} , {#Bloques}, {#Mina
                print(pool);
                print("id\t hash \t saldo");
                for key in usuarios:
                    print(str(usuarios[key]));
            if __name__ == "__main__":
                main()
```

```
La cadena tiene un 1 Bloque, el genésis
Agregando nodos mineros para la blockchain ...
agregados 5 mineros ...
Iniciando la simulación de crecimiento de la blockchain
Fin de la simulación
{Descripcion} , {Dirección} , {Puerto} , {Utilidad} , {#Bloques}, {#Minadores}
nodo minero 0 , 127.0.0.152:60109 , 74630 , 8 , 5
nodo minero 1 , 127.0.0.123:51402 , 193207 , 21 , 5
nodo minero 2 , 127.0.0.112:18794 , 277198 , 30 , 5
nodo minero 3 , 127.0.0.137:48074 , 271195 , 30 , 5
nodo minero 4 , 127.0.0.27:32727 , 90562 , 10 , 5
```

```
id         hash          saldo
0, ae1bc1b04561fac552570296dfe154094dc4df4b6d4970ca81ec9507046375b2 , 114.926
1, 9d5eab463378fe259f33d07590aa6a913a1d099d5a276e353a955cb2da4311f2 , 14.926
2, b4aec9b924aa7f7d83851c90a7f5142be0cec41c7c46cff2fb89bf0bbf055f67 , 14.926
3, 0890bee71d46a465a64625124eee9a424a66908e11ce283ed4afaf5a9551cef0 , 14.926
4, 1206cfe53ef4288f00f4c53f5377601c07bdc95f6a5249b95183cc5db54ad93a , 14.926
5, 5d4a62b58203901f4e7502ccbdd5a2e52a234bb44545549fd5c87edef9202363 , 38.641400000000004
6, 11446965a28709ff7577c812099db7093ae7a31f8aa8ebe2a81d15ee876b0f16 , 38.641400000000004
7, 44b70f5a64febf0f1c47390efaa1d824a5503e3831d37b28b42c63f25813cf17 , 38.641400000000004
8, 06aeb9cf521c8e7114647117303ebf837f50bd39b38104ce5e3371bf54d1db61 , 38.641400000000004
9, 371eb3f89a125644770e3395faa25a50d0f7bf6f9fb15b271649b53640ed1a3b , 38.641400000000004
10, d3e0b66ec1e56b92a5c0b1bd7c57540697da963987253413c40b77636707f13c , 55.439599999999984
11, b29492fe4bf566a54a5ac57bfb49cd6441b5cc3dbb19b36804da13b6a7495032 , 55.439599999999984
12, e8a26469e9a9879b4ef23fb12236e77a66b7e1a55b060694537c39fdca4fec8c , 55.439599999999984
13, 526042ccba454c2762af9fefd0d418254372e3d3c9d8605dfa6217d1c5843a61 , 55.439599999999984
14, 7f74614d41796a901e5dd4fc962da90a80b2e0a7a9f3180d968697845d882e55 , 55.439599999999984
15, c1f330b2fa35be43c2d2e695cc0d73ba2c074247ef4e10a82e4a4eca63307c80 , 54.239000000000004
16, da6a1600010abc6c5a6c9aea97886cf0559fe1ab067110a0dabb3e6c449c3835 , 54.239000000000004
17, 33d772cef882b98fe5495ddc7b1e7b369fb6301372761b36542aa7f2683a0788 , 54.239000000000004
18, 5fafa6531ad8489b778a631289e5d41cc2a6dee01b2d17dab578b9ecc47abdac , 54.239000000000004
19, ce12b6eece7dc590ac7af2b77258ba8c262d5d0b1f89b8aff89914e6eb3a86b9 , 54.239000000000004
20, b252c76b28f15c2fce8bfb7b0153ce87336a0789e16c40fed7faa04df2fc864f , 18.112400000000004
21, 839604719f7a03403bda1bea57e369f0ee226a4aa8a092aee337252b330e7ab8 , 18.112400000000004
22, 18fa0626ea0690666f53c910244389a32c4fa7a13453987b094ddd813ec1904d , 18.112400000000004
23, 192990181fd61e86ffa1e28ec8741a76c1640e816bd75b00e8e7e4499b12c1af , 18.112400000000004
24, 1d918ed33bf67966f2ab8ea55bb3707f2ead718e01b6881c5ec03882587760e1 , 18.112400000000004
```

Clase Simulador

```
In [14]: #!/usr/bin/env python3
         # -*- coding: utf-8 -*-

         from random import randint
         from Usuario import Usuario
         from CadenaBloques import CadenaBloques
         from NodoMinador import NodoMinador
         from PoolMinado import PoolMinado
         from random import randrange
         from random import random
         from random import seed
         from datetime import datetime

         """
         Define la estructura del bloque.
         """
```

```
class Simulador(object):

    def __init__(self, nombre):
        """
        Creación del escenario final de simulación
        :param _nombre Nombre Asignado al escenario
        :param _usuarios Conjunto de usuarios en la simulación
        :param _blockchain Cadena de Bloques
        :param _pool Conjunto de nodos mineros de la cadena
        """
        self._nombre = nombre;
        self._usuarios = {};
        self._blockchain = CadenaBloques();
        self._pool = PoolMinado()

    def crearUsuarios(self, cantidadUsuarios):
        """
        Inicializa el conjunto de usuarios de la cadena de bloques
        :param cantidadUsuarios: Conjunto inicial de Usuarios en la Simulación
        """
        for i in range(cantidadUsuarios):
           temp = Usuario(i);
           temp.setHashUsuario();
           self._usuarios[temp.gethashUsuario()] = temp;

    def distribuirUsuariosMineros(self, maxUsuarios, maxMineros):
        """
        Permite distribuir de manera uniforme los usuarios a los nodos mineros
        :param maxMineros: Conjunto de Nodos mineros de la blockchain
        :param maxUsuarios: Total de usuarios en el sistema
        :return arreglo con los usuarios distribuidos para cada nodo minero
        """
        #Se construye una conjunto aleatorio de nodos mineros
        usuariosNodo = [];
        usuariosMineros = {};
        contador = maxUsuarios//maxMineros - 1;
        aux = 0;
        for key in self._usuarios:
            usuariosMineros[key] = self._usuarios[key];
            if(aux < contador):
                aux += 1;
            else:
                usuariosNodo.append(usuariosMineros);
                aux = 0;
                usuariosMineros = {};
        return usuariosNodo;

    def verUsuarios(self):
```

```
        """
        Consulta el total de usuarios registrados en un momento dado en la cadena de
        :return  Representación en string del conjunto de Usuarios en la Simulación
        """
        a = "";
        for key in self._usuarios:
            a += str(self._usuarios[key])+"\n";
        return a;

    def consultarHashUsuario(self, idUsuario):
        """
        Retorna el Hash de un usuario dado su número de identificación
        :param idUsuario identificador entero de cada usuario en el sistema
        :return Hash de usuario que corresponde al identificador dado
        """
        for key in self._usuarios:
            if(self._usuarios[key]._idUsuario == idUsuario ):
                return self._usuarios[key].gethashUsuario();

    def consultarUsuario(self, hashUsuario):
        """
        Consulta la información de un usuario dado su hash
        :param hashUsuario LLave del diccionario
        :return usuario que corresponde con la llave dada
        """
        return self._usuarios[hashUsuario];

    def inicializarBlockchain(self, maxUsuarios, cantidadMineros):
        """
        Creación del bloque génesis de la cadena y asignación de nodos mineros
        :param cantidadMineros, Total de nodos Mineros para la cadena
        :param maxUsuarios: Total de usuarios en el sistema
        """
        emisor = self.consultarUsuario(self.consultarHashUsuario(0));
        receptor = self.consultarUsuario(self.consultarHashUsuario(0));
        transacciones = [];
        transacciones.append(emisor.enviar(100,receptor.gethashUsuario()));
        self._blockchain.bloqueGenesis(transacciones);
        usuariosNodo = self.distribuirUsuariosMineros(maxUsuarios,cantidadMineros);
        self.creacionMineros(cantidadMineros,usuariosNodo);

    def creacionMineros(self, cantidadMineros, usuariosNodo):
        """
        Adiciona los nodos mineros a la blockchain. Cada nodo minero tendrá asociado
        un grupo de usuarios, quienés recibirán recompensa por su esfuerzo de minado.
        :param cantidadMineros, Total de nodos Mineros para la cadena
        :param usuariosNodo, Distribución de usuarios mineros para cada nodo Minero
        """
```

```
        for m in range(cantidadMineros):
            address = "127.0.0." + str(1+randint(1,250));
            puerto = 1025+randint(1000,10000);
            descripcion = "Minero " + str(m);
            minerito = NodoMinador(address, puerto,descripcion,usuariosNodo[m]);
            minerito.actualizarCadena(self._blockchain);
            self._pool.adicionarMinero(minerito);

    def simular(self, maxTransaccionesPorBloque,totalBloquesCaena, cantidadUsuarios):
        """
        Realiza una simulación de evolución de la blockchain con los parámetros dados
        :param cantidadUsuarios, Total de usuarios activos en la cadena
        :param cantidadMineros, Total de nodos Mineros para la cadena
        :param usuariosNodo, Distribución de usuarios mineros para cada nodo Minero
        """
        transacciones = [];
        emisor = self.consultarHashUsuario(0);
        receptor = self.consultarHashUsuario(1);
        cantidad =  self.consultarUsuario(emisor)._saldo * random();
        transacciones.append(self.consultarUsuario(emisor).enviar(cantidad, self.consu
        self._pool.torneoPorNuevoBloque(transacciones);
        cantidadPosibleReceptores = 1;
        for i in range(1,totalBloquesCaena):
            transacciones = [];
            if cantidadPosibleReceptores == cantidadUsuarios-1:
                cantidadPosibleReceptores = 1;
            #Se construye una conjunto aleatorio de transacciones
            for tr in range(1+randrange(maxTransaccionesPorBloque)):
                if cantidadPosibleReceptores == cantidadUsuarios-1:
                    cantidadPosibleReceptores = 1;
                intemisor = self.seleccionarUsuario(cantidadPosibleReceptores,-1);
                intreceptor = self.seleccionarUsuario(cantidadPosibleReceptores,intemis
                emisor = self.consultarHashUsuario(intemisor);
                receptor = self.consultarHashUsuario(intreceptor);
                cantidad =  self.consultarUsuario(emisor)._saldo * random();
                #print("Cantidad receptores " + str(cantidadPosibleReceptores)+ " rece
                print(".", end="");
                transacciones.append(self.consultarUsuario(emisor).enviar(cantidad, se
                cantidadPosibleReceptores += 1;
            instanteInicial = datetime.now()
            self._pool.torneoPorNuevoBloque(transacciones);
            instanteFinal = datetime.now()
            tiempo = instanteFinal - instanteInicial # Devuelve un objeto timedelta
            segundos = tiempo.microseconds
            print("Torneo " + str(i) + " " + str(segundos) + " microseconds");

    def seleccionarUsuario(self,cantidadPosibleReceptores, diferentea):
        seed();
```

```
            x = randint(0,cantidadPosibleReceptores+1);
            while x == diferentea:
                x = randint(0,cantidadPosibleReceptores+1);
            return x;

        def verNodosMineros(self):
            """
            Consulta el total de nodos mineros registrados en la cadena de bloques
            :return pool de mineros de la blockchain
            """
            return self._pool;
```

Prueba Unitaria Clase Simulador

```
In [15]: #!/usr/bin/env python3
         # -*- coding: utf-8 -*-
         from Simulador import Simulador
         from datetime import datetime

         """
         En esta clase se realizan pruebas de unidad
         """
         class pruebasUnidadSimulador(object):

             def main():
                sm = Simulador("Simulación del comportamiento de una cadena de bloques");
                #cantidadUsuarios = int(input("Cantidad de usuarios en la simulación: "));
                cantidadUsuarios = 20;
                sm.crearUsuarios(cantidadUsuarios);
                #cantidadMineros = int(input("Cantidad de Mineros de la blockchain: "));
                cantidadMineros = 5;
                #maxTransaccionesPorBloque = int(input("Maxima cantidad de transacciones por B
                maxTransaccionesPorBloque = 5;
                #totalBloquesCaena = int(input("Cantidad de Bloques en la Cadena: "));
                totalBloquesCaena = 1000;
                print("Creando la cadena de bloques...Adicionando Bloque Génesis...");
                print("Hash usuario génesis: " + str(sm.consultarHashUsuario(0)));
                sm.inicializarBlockchain(cantidadUsuarios,cantidadMineros);
                print("Imprimiendo Bloque Génesis...")
                print(sm._blockchain.getCadenaSerializada());

                instanteInicial = datetime.now()
                sm.simular(maxTransaccionesPorBloque,totalBloquesCaena,cantidadUsuarios);
                instanteFinal = datetime.now()
                tiempo = instanteFinal - instanteInicial # Devuelve un objeto timedelta
                segundos = tiempo.seconds;
                print("\nFin de la simulación: demoro " + str(segundos) + " segundos");
                print("Estado de los Nodos Mineros...");
```

```
            print("{} , {} , {} , {}, {}".format(str("Descripción"),str("Direccion"), str("
            print(sm.verNodosMineros());
            print("Estado de los Usuarios")
            print("{}, {} , {} ".format(str("#id"),str("Direccion Hash del Usario"), str("S
            print(sm.verUsuarios());
        if __name__ == "__main__":
            main()
```

```
Creando la cadena de bloques...Adicionando Bloque Génesis...
Hash usuario génesis: ae1bc1b04561fac552570296dfe154094dc4df4b6d4970ca81ec9507046375b2
Imprimiendo Bloque Génesis...
0 , 2 , 5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9  , 1529248135.5695615
Fin Bloque 0.

...
Fin de la simulación: demoro 107 segundos
Estado de los Nodos Mineros...
Descripción , Direccion , Utilidad , Bloques, Usuarios
Minero 0 , 127.0.0.20:2999 , 911962 , 100 , 4
Minero 1 , 127.0.0.132:9888 , 1955814 , 211 , 4
Minero 2 , 127.0.0.132:3869 , 2640455 , 282 , 4
Minero 3 , 127.0.0.48:7341 , 2844724 , 305 , 4
Minero 4 , 127.0.0.185:6264 , 958680 , 102 , 4

Estado de los Usuarios
#id, Direccion Hash del Usario , Saldo
0, ae1bc1b04561fac552570296dfe154094dc4df4b6d4970ca81ec9507046375b2 , 43.90571953228023
1, 9d5eab463378fe259f33d07590aa6a913a1d099d5a276e353a955cb2da4311f2 , 170.85615671626476
2, b4aec9b924aa7f7d83851c90a7f5142be0cec41c7c46cff2fb89bf0bbf055f67 , 83.2765994522879
3, 0890bee71d46a465a64625124eee9a424a66908e11ce283ed4afaf5a9551cef0 , 1623.9545723143294
4, 1206cfe53ef4288f00f4c53f5377601c07bdc95f6a5249b95183cc5db54ad93a , 125.85689736016461
5, 5d4a62b58203901f4e7502ccbdd5a2e52a234bb44545549fd5c87edef9202363 , 368.81631273914695
6, 11446965a28709ff7577c812099db7093ae7a31f8aa8ebe2a81d15ee876b0f16 , 193.74265994235944
7, 44b70f5a64febf0f1c47390efaa1d824a5503e3831d37b28b42c63f25813cf17 , 745.5559940692958
8, 06aeb9cf521c8e7114647117303ebf837f50bd39b38104ce5e3371bf54d1db61 , 290.65435766067515
9, 371eb3f89a125644770e3395faa25a50d0f7bf6f9fb15b271649b53640ed1a3b , 839.5940516116573
10, d3e0b66ec1e56b92a5c0b1bd7c57540697da963987253413c40b77636707f13c , 40.95932113167698
11, b29492fe4bf566a54a5ac57bfb49cd6441b5cc3dbb19b36804da13b6a7495032 , 3083.063430232901
12, e8a26469e9a9879b4ef23fb12236e77a66b7e1a55b060694537c39fdca4fec8c , 73.9606016876291
13, 526042ccba454c2762af9fefd0d418254372e3d3c9d8605dfa6217d1c5843a61 , 606.9877366153947
14, 7f74614d41796a901e5dd4fc962da90a80b2e0a7a9f3180d968697845d882e55 , 313.7516847236654
15, c1f330b2fa35be43c2d2e695cc0d73ba2c074247ef4e10a82e4a4eca63307c80 , 21.880188867958633
16, da6a1600010abc6c5a6c9aea97886cf0559fe1ab067110a0dabb3e6c449c3835 , 603.3334429737754
17, 33d772cef882b98fe5495ddc7b1e7b369fb6301372761b36542aa7f2683a0788 , 4.072810289791246
18, 5fafa6531ad8489b778a631289e5d41cc2a6dee01b2d17dab578b9ecc47abdac , 88.3769476493061
19, ce12b6eece7dc590ac7af2b77258ba8c262d5d0b1f89b8aff89914e6eb3a86b9 , 89.03551442943997
```

In [ ]: