

# Cita Previa con Blockchain - Prueba de Concepto, En Lenguaje Python

**ADAPTADO PARA INCLUIR LOS DATOS DEL SISTEMA DE CITA PREVIA DEL AYUNTAMIENTO DE MADRID EN UNA CADENA DE BLOQUES.**

**BASADO EN: Build Your Own Blockchain - The Basics**

**ENLACE:**

<http://ecomunising.com/build-your-own-blockchain> (<http://ecomunising.com/build-your-own-blockchain>)

This tutorial will walk you through the basics of how to build a blockchain from scratch. Focusing on the details of a concrete example will provide a deeper understanding of the strengths and limitations of blockchains. For a higher-level overview, I'd recommend [this excellent article from BitsOnBlocks](https://bitsonblocks.net/2015/09/09/a-gentle-introduction-to-blockchain-technology/) (<https://bitsonblocks.net/2015/09/09/a-gentle-introduction-to-blockchain-technology/>).

## DATASET CITA PREVIA:

- PARA ESTE EJEMPLO USO EL DATASET DEL AYUNTAMIENTO SOBRE LOS DATOS DE CITA PREVIA, CORRESPONDIENTES A ENERO DEL 2017
- EL FORMATO DEL MISMO ES EL SIGUIENTE Y DEBAJO ESTA LA PRIMERA LINEA DE DATOS:

**CANAL / DETALLE\_CANAL / FECHA\_CONCERTACION / OFICINA\_CITA / FECHA\_ATENCION / TAREA / ATENDIDA**

Internet / Fijo / 05/04/2017 / OAC CIUDAD LINEAL / 11/04/2017 / PADRON / No

## NOTAS SOBRE EL DATASET:

**NOMBRE DEL FICHERO:** datos\_cita\_previa\_201711.csv

ESTE FICHERO CONTIENE TODOS LOS DATOS CORRESPONDIENTES A 2017, SUBIDOS HASTA LA FECHA DE DESCARGA  
ENLACE A LA PAGINA GENERAL:

<http://datos.madrid.es/sites/v/index.jsp?vgnextoid=3551a6fb1326a410VgnVCM1000000b205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aF>  
<http://datos.madrid.es/sites/v/index.jsp?vgnextoid=3551a6fb1326a410VgnVCM1000000b205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aF>

## A CONTINUACION, AÑADO UNAS PEQUEÑAS NOTAS DE ANÁLISIS DEL DATASET:

**DATASET:** datos\_cita\_previa\_201711.csv

**CABECERA:** 7 CAMPOS EN TOTAL

**CANAL; DETALLE\_CANAL; FECHA\_CONCERTACION; OFICINA\_CITA; FECHA\_ATENCION; TAREA; ATENDIDA**

**ANALISIS:** 3 BLOQUES EN TOTAL:

**TOTAL DE CONSULTAS: 997.043**

2 -> 288.362	INTERNET (MOVIL O FIJO)
TOTAL: 288.360	
288363 -> 751176	PRESENCIAL
TOTAL: 462.813	
751177 -> 997044	TELEFONICO 010
TOTAL: 245.867	

## DATASET REDUCIDO, A PARTIR DE LOS DATOS ORIGINALES:

Los datos corresponden a las consultas de cita previa gestionadas durante el mes de Enero de 2017.

El dataset original está dividido en tres bloques principales. El elemento separador es es tabulador.

Hemos creado un subset con los 100 primeros datos de cada bloque 1º)  
datos\_cita\_previa\_201711\_SEL\_100prim\_cada\_bloque01.csv

Para las pruebas inciales, he creado un subset sólo con 10 registros: dataset\_7columnas.csv:

A CONTINUACION, UN CODIGO PARA CARGAR Y MOSTRAR EL DATASET DE DOS COLUMNAS:

\* NOTA: EN LA TABLA, EL CARACTER TABULADOR SE REPRESENTA COMO \t

In [2]:	<pre>import numpy as np import pandas as pd  # PARA QUE LA LECTURA DEL FICHERO FUNCIONE EN TODOS LOS LADOS, TIENE QUE ESTAR EN LA MISMA CARPETA QUE # columnas = pd.read_csv("dataset_7columnas.csv") # SUBIDO A UN SERVIDOR: columnas = pd.read_csv("http://datamad.1leo.net/citaprevia/dataset_7columnas.csv")</pre>																						
In [3]:	<pre># print (columnas) columnas</pre>																						
Out[3]:	<table border="1"> <thead> <tr> <th></th> <th>CANAL DETALLE_CANAL FECHA_CONCERTACION OFICINA_CITA FECHA_ATENCION TAREA ATENDIDA</th> </tr> </thead> <tbody> <tr> <td>0</td><td>Internet\tFijo\t05/04/2017\tOAC CIUDAD LINEAL\...</td></tr> <tr> <td>1</td><td>Internet\tFijo\t24/08/2017\tOAC FUENCARRAL EL ...</td></tr> <tr> <td>2</td><td>Internet\tFijo\t05/04/2017\tOFICINA DE GESTION...</td></tr> <tr> <td>3</td><td>Internet\tFijo\t27/02/2017\tCMS INTERNACIONAL\...</td></tr> <tr> <td>4</td><td>Presencial\tOAC VILLA DE VALLECA\t21/09/2017\t...</td></tr> <tr> <td>5</td><td>Presencial\tOAC MONCLOA ARAVACA\t14/06/2017\tO...</td></tr> <tr> <td>6</td><td>Presencial\tOAC SAN BLAS-CANILLE\t03/03/2017\t...</td></tr> <tr> <td>7</td><td>Telefonico\t10\t17/10/2017\tOAIC RAMON POWER 2...</td></tr> <tr> <td>8</td><td>Telefonico\t10\t02/08/2017\tOAC VICALVARO\t07/...</td></tr> <tr> <td>9</td><td>Telefonico\t10\t28/09/2017\tAGENCIA DE ACTIVID...</td></tr> </tbody> </table>		CANAL DETALLE_CANAL FECHA_CONCERTACION OFICINA_CITA FECHA_ATENCION TAREA ATENDIDA	0	Internet\tFijo\t05/04/2017\tOAC CIUDAD LINEAL\...	1	Internet\tFijo\t24/08/2017\tOAC FUENCARRAL EL ...	2	Internet\tFijo\t05/04/2017\tOFICINA DE GESTION...	3	Internet\tFijo\t27/02/2017\tCMS INTERNACIONAL\...	4	Presencial\tOAC VILLA DE VALLECA\t21/09/2017\t...	5	Presencial\tOAC MONCLOA ARAVACA\t14/06/2017\tO...	6	Presencial\tOAC SAN BLAS-CANILLE\t03/03/2017\t...	7	Telefonico\t10\t17/10/2017\tOAIC RAMON POWER 2...	8	Telefonico\t10\t02/08/2017\tOAC VICALVARO\t07/...	9	Telefonico\t10\t28/09/2017\tAGENCIA DE ACTIVID...
	CANAL DETALLE_CANAL FECHA_CONCERTACION OFICINA_CITA FECHA_ATENCION TAREA ATENDIDA																						
0	Internet\tFijo\t05/04/2017\tOAC CIUDAD LINEAL\...																						
1	Internet\tFijo\t24/08/2017\tOAC FUENCARRAL EL ...																						
2	Internet\tFijo\t05/04/2017\tOFICINA DE GESTION...																						
3	Internet\tFijo\t27/02/2017\tCMS INTERNACIONAL\...																						
4	Presencial\tOAC VILLA DE VALLECA\t21/09/2017\t...																						
5	Presencial\tOAC MONCLOA ARAVACA\t14/06/2017\tO...																						
6	Presencial\tOAC SAN BLAS-CANILLE\t03/03/2017\t...																						
7	Telefonico\t10\t17/10/2017\tOAIC RAMON POWER 2...																						
8	Telefonico\t10\t02/08/2017\tOAC VICALVARO\t07/...																						
9	Telefonico\t10\t28/09/2017\tAGENCIA DE ACTIVID...																						

**NOTA SOBRE JUPYTER NOTEBOOK: PARA EJECUTAR UN BLOQUE (CELDA) DE CODIGO, HAY QUE PULSAR SHIFT+ENTER**

## TRANSACCIONES, VALIDACIÓN Y ACTUALIZACION DEL ESTADO DEL SISTEMA:

- UN BLOCKCHAIN ES UNA BASE DE DATOS DISTRIBUIDA, CON UNA SERIE DE REGLAS PARA VERIFICAR NUEVAS ADICIONES A LA BASE DE DATOS
  - VAMOS A CREAR BLOQUES QUE INCLUYAN UN CIERTO NÚMERO DE REGISTROS, PROCEDENTES DEL DATASET DE CITA PREVIA
  - DEBEMOS CREAR LA PRIMERA VEZ UN BLOQUE INICIAL, LLAMADO GENESIS
  - UNA VEZ CREADO UN BLOQUE NUEVO, LO ENCADENAMOS AL ANTERIOR
  - VAMOS A USAR UNA FUNCION HASH (CRIPTOGRÁFICA) PARA CREAR UNA HUELLA (FINGERPRINT) DE CADA REGISTRO AÑADIDO. ESTA FUNCION HASH ENLAZA CADA UNO DE NUESTROS BLOQUES CON CADA UNO DE LOS OTROS.
    - PARA QUE SEA MAS FACIL DE USAR, DEFINIMOS UNA FUNCIÓN DE AYUDA QUE ENCAPSULA LA FUNCIÓN HASH DE PYTHON QUE ESTAMOS USANDO.

## FUNCION DE AYUDA:

```
In [5]: import hashlib, json

def hashMe(msg=""):
    # For convenience, this is a helper function that wraps our hashing algorithm
    if type(msg)!=str:
        msg = json.dumps(msg,sort_keys=True) # If we don't sort keys, we can't guarantee repeatability

    return unicode(hashlib.sha256(msg).hexdigest(),'utf-8')
```

PRUEBA DE LA FUNCION: hashMe ("algo para encriptar")

\* FUNCIONA PARA STRINGS Y PARA NUMEROS

```
In [6]: hashMe ("algo para encriptar")
```

```
Out[6]: u'5f2d2ceababb317f87b4208ade0216767244acc56fca0fa63b46c73b767118d'
```

### ESTA FUNCION ES LA ENCARGADA LOS DATOS DEL DATASET DE CITA PREVIA

- SIETE COLUMNAS: return {u'Canal': "Presencial", u'DetalleCanal': "OAC VILLA DE VALLECA", u'FechaConcer': "21/09/2017", u'OficinaCita': "OMIC MORATALAZ", u'FechaAtt': "26/09/2017", u'Tarea': "INFORMACION Y NUEVAS RECLAMACIONES", u'Atendida': "Si"}
  - USANDO DATOS DIRECTAMENTE PUESTOS EN EL CODIGO

```
In [34]: def makeTransactionEd(maxValue=10):
    # LO QUE TIENE QUE DEVOLVER SON LOS DATOS DEL DATASET
    # EN PRINCIPIO, LO RESUELVO PONIENDO LOS DATOS DIRECTAMENTE
    # QUEDA PENDIENTE HACER QUE LO LEA DESDE UN ARCHIVO .CSV
    # A 7 COLUMNAS:
    return {u'Canal': "Presencial", u'DetalleCanal': "OAC VILLA DE VALLECA", u'FechaConcer': "21/09/2017", u'FechaAtt': "26/09/2017", u'OficinaCita': "OMIC MORATALAZ", u'Tarea': "INFORMACION Y NUEVAS RECLAMACIONES", u'Atendida': "Si"}
```

```
In [9]: makeTransactionEd(1)
```

```
Out[9]: {u'Atendida': 'Si',
         u'Canal': 'Presencial',
         u'DetalleCanal': 'OAC VILLA DE VALLECA',
         u'FechaAtt': '26/09/2017',
         u'FechaConcer': '21/09/2017',
         u'OficinaCita': 'OMIC MORATALAZ',
         u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}
```

## PENDIENTE:

\* Crear un código que lea desde un archivo .csv y lo prepare para poder crear una cadena de bloques  
 \*

## BUCLE DE TRANSACCIONES

AHORA VAMOS A CREAR UN CONJUNTO MAYOR DE TRANSACCIONES Y LOS DIVIDIMOS ENTRE BLOQUES

```
In [11]: txnBuffer = [makeTransactionEd() for i in range(10)]
```

### IMPRIMIMOS EN PANTALLA EL RESULTADO DEL BUCLE:

In [12]: txnBuffer

# DUDA, BORRAR???

## ESTE CODIGO SIRVE PARA ACTUALIZAR EL ESTADO DE LAS TRANSACCIONES

```
In [13]: def updateState(txn, state):
    # Inputs: txn, state: dictionaries keyed with account names, holding numeric values for transfer amounts
    # Returns: Updated state, with additional users added to state if necessary
    # NOTE: This does not validate the transaction- just updates the state!

    # If the transaction is valid, then update the state
    state = state.copy() # As dictionaries are mutable, let's avoid any confusion by creating a working copy
    for key in txn:
        if key in state.keys():
            state[key] += txn[key]
        else:
            state[key] = txn[key]
    return state
```

## DUDA, BORRAR???

### ESTE CODIGO SIRVE PARA VALIDAR LAS TRANSACCIONES

```
In [14]: def isValidTxn(txn,state):
    # Assume that the transaction is a dictionary keyed by account names

    # Check that the sum of the deposits and withdrawals is 0
    if sum(txn.values()) is not 0:
        return False

    # Check that the transaction does not cause an overdraft
    for key in txn.keys():
        if key in state.keys():
            acctBalance = state[key]
        else:
            acctBalance = 0
        if (acctBalance + txn[key]) < 0:
            return False

    return True
```

## CONSTRUYENDO EL BLOCKCHAIN:

### DESDE LAS TRANSACCIONES A LOS BLOQUES

¡YA ESTAMOS PREPARADOS PARA COMENZAR A CONSTRUIR NUESTRO BLOCKCHAIN!

POR AHORA, NO HAY NADA EN EL BLOCKCHAIN, PERO PODEMOS CONSEGUIR QUE TODO COMIENCE DEFINIENDO EL BLOQUE GENESIS (EL PRIMER BLOQUE DEL SISTEMA).

### BLOQUE GENESIS:

COMO EL BLOQUE GENESIS NO ESTA ENLAZADO A NINGUN BLOQUE PREVIO, TIENE QUE SER TRATADO DE UN MODO DIFERENTE. PODEMOS AJUSTAR EL ESTADO DEL SISTEMA ARBITRARIAMENTE.

```
In [16]: # ESTA LINEA SOLO DEFINE EL ESTADO INICIAL, PARA GENERAR EL BLOQUE GENESIS:
state = {u'Canal': "Presencial", u'DetalleCanal': "OAC VILLA DE VALLECA", u'FechaConcer': "21/09/2017"}
genesisBlockTxns = [state]
genesisBlockContents = {u'blockNumber':0,u'parentHash':None,u'txnCount':1,u'txns':genesisBlockTxns}
genesisHash = hashMe( genesisBlockContents )
genesisBlock = {u'hash':genesisHash,u'contents':genesisBlockContents}
genesisBlockStr = json.dumps(genesisBlock, sort_keys=True)
```

IMPRIMIMOS EL PRIMER ELEMENTO (GENESIS) A PARTIR DEL CUAL, TODOS LOS SIGUIENTES TIENEN QUE ESTAR ENLAZADOS.

In [17]: `print (state)`

```
{u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}
```

## A CONTINUACION, AÑADIMOS EL PRIMER BLOQUE (GENESIS) A LA CADENA DE BLOQUES (VARIABLE CHAIN)

In [18]: `chain = [genesisBlock]`

### NOTAS SOBRE EL BLOCKCHAIN Y SUS FUNCIONES:

- SI PONGO UNA SENTENCIA COMO: `print (chain)` IMPRIME POR PANTALLA EL CONTENIDO DEL BLOCKCHAIN:
  - SI SOLO SE HA CREADO EL PRIMER BLOQUE (GENESIS), SOLO SE IMPRIME UNO
  - PERO SI YA HAY HECHAS UNAS CUANTAS TRANSACCIONES, AL EJECUTAR `print (chain)`, SALEN TODAS LAS TRANSACCIONES EJECUTADAS

### IMPRIMIMOS A CONTINUACION EL CONTENIDO DE LAS DISTINTAS VARIABLES DEL BLOQUE GENESIS

In [19]: `print (chain)`

```
[{u'hash': u'd5dc13b6c9658c60b67efcebf54eb2a7a896f5e69532774d8fd6da91c70b9aab', u'contents': {u'txns': [{u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}], u'parentHash': None, u'txnCount': 1, u'blockNumber': 0}]
```

In [20]: `print (genesisBlock)`

```
{u'hash': u'd5dc13b6c9658c60b67efcebf54eb2a7a896f5e69532774d8fd6da91c70b9aab', u'contents': {u'txns': [{u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}], u'parentHash': None, u'txnCount': 1, u'blockNumber': 0}}
```

In [21]: `print (genesisBlockStr)`

```
{"contents": {"blockNumber": 0, "parentHash": null, "txnCount": 1, "txns": [{"Atendida": "Si", "Canal": "Presencial", "DetalleCanal": "OAC VILLA DE VALLECA", "FechaAtt": "26/09/2017", "FechaConcer": "21/09/2017", "OficinaCita": "OMIC MORATALAZ", "Tarea": "INFORMACION Y NUEVAS RECLAMACIONES"}]}, "hash": "d5dc13b6c9658c60b67efcebf54eb2a7a896f5e69532774d8fd6da91c70b9aab"}
```

## REPASAR:

A CONTINUACION PRUEBO ALGUNOS TROZOS DE CODIGO Y A IMPRIMIR ALGUNAS VARIABLES, PARA VER LOS CONTENIDOS:

In [25]: `genesisBlockStr1 = json.dumps(genesisBlock, sort_keys=False)`

In [ ]:

In [26]: `print (genesisBlockStr1)`

```
{"hash": "d5dc13b6c9658c60b67efcebf54eb2a7a896f5e69532774d8fd6da91c70b9aab", "contents": {"txns": [{"Canal": "Presencial", "OficinaCita": "OMIC MORATALAZ", "DetalleCanal": "OAC VILLA DE VALLECA", "FechaAtt": "26/09/2017", "Atendida": "Si", "FechaConcer": "21/09/2017", "Tarea": "INFORMACION Y NUEVAS RECLAMACIONES"}]}, "parentHash": null, "txnCount": 1, "blockNumber": 0}}
```

In [27]: `print (genesisBlockTxns)`

```
[{u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}]
```

```
In [28]: print (state)
```

```
{u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA',
u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y
NUEVAS RECLAMACIONES'}
```

```
In [29]: print (genesisHash)
```

```
d5dc13b6c9658c60b67efcebf54eb2a7a896f5e69532774d8fd6da91c70b9aab
```

```
In [30]: print (genesisBlockContents)
```

```
{u'txns': [{u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE
VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INF
ORMACION Y NUEVAS RECLAMACIONES'}], u'parentHash': None, u'txnCount': 1, u'blockNumber': 0}
```

## EL CODIGO A CONTINUACION ES EL BUCLE QUE GENERA EL BLOQUE GENESIS

```
In [31]: def makeBlock(txns,chain):
    parentBlock = chain[-1]
    parentHash  = parentBlock[u'hash']
    blockNumber = parentBlock[u'contents'][u'blockNumber'] + 1
    txnCount   = len(txns)
    blockContents = {u'blockNumber':blockNumber,u'parentHash':parentHash,
                     u'txnCount':len(txns), 'txns':txns}
    blockHash = hashMe( blockContents )
    block = {u'hash':blockHash,u'contents':blockContents}

    return blockprint (genesisBlockContents)
```

```
In [32]: print (chain)
```

```
[{u'hash': u'd5dc13b6c9658c60b67efcebf54eb2a7a896f5e69532774d8fd6da91c70b9aab', u'contents': {u'txn
s': [{u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLE
CA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMAC
ION Y NUEVAS RECLAMACIONES'}], u'parentHash': None, u'txnCount': 1, u'blockNumber': 0}}]
```

## REPASAR: LA VARIABLE TXNS (PUEDE SER) EL NÚMERO DE TRANSACCIONES

```
In [35]: txns = 1
print (txns)
```

```
1
```

```
In [38]: makeBlock (txns, chain)
```

```
-----
TypeError                                                 Traceback (most recent call last)
<ipython-input-38-12a47a69b747> in <module>()
----> 1 makeBlock (txns, chain)

<ipython-input-31-555e7b6a7f60> in makeBlock(txns, chain)
      3     parentHash  = parentBlock[u'hash']
      4     blockNumber = parentBlock[u'contents'][u'blockNumber'] + 1
----> 5     txnCount   = len(txns)
      6     blockContents = {u'blockNumber':blockNumber,u'parentHash':parentHash,
      7                         u'txnCount':len(txns), 'txns':txns}
```

```
TypeError: object of type 'int' has no len()
```

In [39]: `print (blockprint)`

```
NameError                                 Traceback (most recent call last)
<ipython-input-39-a7bdb03cf25> in <module>()
      1 print (blockprint)

NameError: name 'blockprint' is not defined
```

In [40]: `print (txns)`

```
1
```

In [41]: `print "chain: "; (chain)
parentBlock = chain[-1]
print "parentBlock: "; (parentBlock)`

```
chain:
parentBlock:
```

Out[41]: {u'contents': {u'blockNumber': 0,
 u'parentHash': None,
 u'txnCount': 1,
 u'txns': [{u'Atendida': 'Si',
 u'Canal': 'Presencial',
 u'DetalleCanal': 'OAC VILLA DE VALLECA',
 u'FechaAtt': '26/09/2017',
 u'FechaConcer': '21/09/2017',
 u'OficinaCita': 'OMIC MORATALAZ',
 u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}]},
 u'hash': u'd5dc13b6c9658c60b67efcebf54eb2a7a896f5e69532774d8fd6da91c70b9aab'}

## EL CODIGO A CONTINUACION ES LA RUTINA QUE GENERA EL BLOQUE

In [42]: `def makeBlock(txns,chain):
parentBlock = chain[-1]
parentHash = parentBlock[u'hash']
blockNumber = parentBlock[u'contents'][u'blockNumber'] + 1
txncount = len(txns)
blockContents = {u'blockNumber':blockNumber,u'parentHash':parentHash,
 u'txnCount':len(txns),'txns':txns}
blockHash = hashMe( blockContents )
block = {u'hash':blockHash,u'contents':blockContents}

return block`

## EL CODIGO A CONTINUACION SON PRUEBAS DE IMPRESION DE DIVERSAS VARIABLES DEL BLOQUE

In [43]: `blockNumber = parentBlock[u'contents'][u'blockNumber'] + 1
print (blockNumber)
# NO FUNCIONA, PORQUE NECESITA UN VALOR PARA txns
# txncount = Len(txns)
# print (txncount)`

```
1
```

## EL CODIGO A CONTINUACION SON PRUEBAS DE ASIGNACION DE VALORES DE DIVERSAS VARIABLES DEL BLOQUE

In [44]: `txns = "hola" # Le doy un valor para que no de error
parentHash = parentBlock[u'hash']
blockContents = {u'blockNumber':blockNumber,u'parentHash':parentHash,
 u'txnCount':len(txns),'txns':txns}
print (blockContents)

{'txns': 'hola', u'parentHash': u'd5dc13b6c9658c60b67efcebf54eb2a7a896f5e69532774d8fd6da91c70b9aab',
 u'txnCount': 4, u'blockNumber': 1}`

## A CONTINUACION EL CODIGO CON EL BUCLE WHILE PARA CONSTRUIR LOS BLOQUES

USEMOS ESTE CODIGO PARA PROCESAR NUESTRO BUFFER DE TRANSACCIONES Y CREAR UN CONJUNTO DE TRANSACCIONES:

\* LO DIVIDO EN VARIAS PARTES PARA HACER LAS PRUEBAS:

```
In [45]: # IMPRESION DE PRUEBA DE ESTAS VARIABLES:
print "txnBuffer: "; (txnBuffer)
print "bufferStartSize: "; (bufferStartSize)
# ERROR: NameError: name 'bufferStartSize' is not defined

txnBuffer:
bufferStartSize:

-----
NameError Traceback (most recent call last)
<ipython-input-45-bb56bcd9abf2> in <module>()
  1 # IMPRESION DE PRUEBA DE ESTAS VARIABLES:
  2 print "txnBuffer: "; (txnBuffer)
----> 3 print "bufferStartSize: "; (bufferStartSize)
  4 # ERROR: NameError: name 'bufferStartSize' is not defined

NameError: name 'bufferStartSize' is not defined
```

```
In [46]: blockSizeLimit = 11 # Arbitrary number of transactions per block-
          # this is chosen by the block miner, and can vary between blocks!
          # CAMBIO POR UN NUMERO MAYOR PARA QUE HAYA MAS BLOQUES
# INCLUYO TAMBIÉN VALORES FICTICIOS PARA LAS VARIABLES QUE SE GENERAN CUANDO SE COMPRUEBA LA VALIDEZ DE
isValidTxn = True # Booleano
validTxn = True # Booleano
```

```
In [47]: while len(txnBuffer) > 0:
    bufferStartSize = len(txnBuffer)
    # IMPRESION DE PRUEBA DE ESTAS VARIABLES:
    print "txnBuffer: "; (txnBuffer)
    print "bufferStartSize: "; (bufferStartSize)

    ## Gather a set of valid transactions for inclusion
    txnList = []
    # ESTE ES EL BUCLE INTERNO
    while (len(txnBuffer) > 0) & (len(txnList) < blockSizeLimit):
        newTxn = txnBuffer.pop()
        # ANULO LA COMPROBACION DE INVALIDEZ
        # validTxn = isValidTxn(newTxn,state) # This will return False if txn is invalid
        validTxn = True

        if validTxn:           # If we got a valid state, not 'False'
            txnList.append(newTxn)
            state = updateState(newTxn,state)
        else:
            print("ignored transaction")
            sys.stdout.flush()
            continue # This was an invalid transaction; ignore it and move on

    ## Make a block
    myBlock = makeBlock(txnList,chain)
    chain.append(myBlock)
```

txnBuffer:  
bufferStartSize:

```
In [48]: print "myBlock:", (myBlock)
```

```
myBlock: {u'hash': u'6b43a55115cb29dcde2d6af7a8bac3a805ec007a59efb4db1c299fc0ed0999f1', u'contents': {'txns': [{u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}, {u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}, {u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}, {u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}, {u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}, {u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}, {u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}, {u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}, {u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}, {u'Canal': 'Presencial', u'OficinaCita': 'OMIC MORATALAZ', u'DetalleCanal': 'OAC VILLA DE VALLECA', u'FechaAtt': '26/09/2017', u'Atendida': 'Si', u'FechaConcer': '21/09/2017', u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}], u'parentHash': u'd5dc13b6c9658c60b67efcebf54eb2a7a896f5e69532774d8fd6da91c70b9aab', u'txnCount': 10, u'blockNumber': 1}}
```

```
In [49]: print "len(txnBuffer):", len(txnBuffer)
```

```
len(txnBuffer): 0
```

```
In [50]: print "len(txnBuffer):", len(txnBuffer)
print "txnBuffer:", txnBuffer
print
print "txnList:", txnList
print
print "newTxn:", newTxn
print
print "isValidTxn:", isValidTxn
print "validTxn:", validTxn
print
print "newTxn:", newTxn
print
print "state:", state
```

## **COMENTARIOS DEL CODIGO ANTERIOR:**

- = 5 NUMERO ARBITRARIO DE TRANSACCIONES POR BLOQUE
  - ESTE VALOR LO ESCOJE EL MINERO DE BLOQUES Y PUEDE VARIAR ENTRE BLOQUES
  - SE REUNEN UN CONJUNTO DE TRANSACCIONES PARA SU INCLUSION
  - ESTA LINEA DEVUELVE FALSO SI  $txn$  ES INVALIDO

- SI CONSEGUIMOS UN ESTADO VALIDO, DISTINTO DE FALSO
- ES UNA TRANSACCION INVALIDA, IGNORALA Y SIGUE
- CONSTRUYE UN BLOQUE

A CONTINUACION, IMPRIMIMOS BLOQUES DE LA CADENA POR SEPARADO, SEGÚN SU NÚMERO DE ORDEN

```
In [51]: chain[0]
```

```
Out[51]: {u'contents': {u'blockNumber': 0,
   u'parentHash': None,
   u'txnCount': 1,
   u'txns': [{u'Atendida': 'Si',
      u'Canal': 'Presencial',
      u'DetalleCanal': 'OAC VILLA DE VALLECA',
      u'FechaAtt': '26/09/2017',
      u'FechaConcer': '21/09/2017',
      u'OficinaCita': 'OMIC MORATALAZ',
      u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}]},
   u'hash': u'd5dc13b6c9658c60b67efcebf54eb2a7a896f5e69532774d8fd6da91c70b9aab'}
```

```
In [52]: chain[1]
```

```
Out[52]: {u'contents': {u'blockNumber': 1,
   u'parentHash': u'd5dc13b6c9658c60b67efcebf54eb2a7a896f5e69532774d8fd6da91c70b9aab',
   u'txnCount': 10,
   'txns': [{u'Atendida': 'Si',
      u'Canal': 'Presencial',
      u'DetalleCanal': 'OAC VILLA DE VALLECA',
      u'FechaAtt': '26/09/2017',
      u'FechaConcer': '21/09/2017',
      u'OficinaCita': 'OMIC MORATALAZ',
      u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'},
    {u'Atendida': 'Si',
      u'Canal': 'Presencial',
      u'DetalleCanal': 'OAC VILLA DE VALLECA',
      u'FechaAtt': '26/09/2017',
      u'FechaConcer': '21/09/2017',
      u'OficinaCita': 'OMIC MORATALAZ',
      u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'},
    {u'Atendida': 'Si',
      u'Canal': 'Presencial',
      u'DetalleCanal': 'OAC VILLA DE VALLECA',
      u'FechaAtt': '26/09/2017',
      u'FechaConcer': '21/09/2017',
      u'OficinaCita': 'OMIC MORATALAZ',
      u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'},
    {u'Atendida': 'Si',
      u'Canal': 'Presencial',
      u'DetalleCanal': 'OAC VILLA DE VALLECA',
      u'FechaAtt': '26/09/2017',
      u'FechaConcer': '21/09/2017',
      u'OficinaCita': 'OMIC MORATALAZ',
      u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'},
    {u'Atendida': 'Si',
      u'Canal': 'Presencial',
      u'DetalleCanal': 'OAC VILLA DE VALLECA',
      u'FechaAtt': '26/09/2017',
      u'FechaConcer': '21/09/2017',
      u'OficinaCita': 'OMIC MORATALAZ',
      u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'},
    {u'Atendida': 'Si',
      u'Canal': 'Presencial',
      u'DetalleCanal': 'OAC VILLA DE VALLECA',
      u'FechaAtt': '26/09/2017',
      u'FechaConcer': '21/09/2017',
      u'OficinaCita': 'OMIC MORATALAZ',
      u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'},
    {u'Atendida': 'Si',
      u'Canal': 'Presencial',
      u'DetalleCanal': 'OAC VILLA DE VALLECA',
      u'FechaAtt': '26/09/2017',
      u'FechaConcer': '21/09/2017',
      u'OficinaCita': 'OMIC MORATALAZ',
      u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'},
    {u'Atendida': 'Si',
      u'Canal': 'Presencial',
      u'DetalleCanal': 'OAC VILLA DE VALLECA',
      u'FechaAtt': '26/09/2017',
      u'FechaConcer': '21/09/2017',
      u'OficinaCita': 'OMIC MORATALAZ',
      u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'},
    {u'Atendida': 'Si',
      u'Canal': 'Presencial',
      u'DetalleCanal': 'OAC VILLA DE VALLECA',
      u'FechaAtt': '26/09/2017',
      u'FechaConcer': '21/09/2017',
      u'OficinaCita': 'OMIC MORATALAZ',
      u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'},
    {u'Atendida': 'Si',
      u'Canal': 'Presencial',
      u'DetalleCanal': 'OAC VILLA DE VALLECA',
      u'FechaAtt': '26/09/2017',
      u'FechaConcer': '21/09/2017',
      u'OficinaCita': 'OMIC MORATALAZ',
      u'Tarea': 'INFORMACION Y NUEVAS RECLAMACIONES'}]}},
   u'hash': u'6b43a55115cb29dcde2d6af7a8bac3a805ec007a59efb4db1c299fc0ed0999f1'}
```

ESTE BLOQUE DE CODIGO LO HE AÑADIDO YO Y FUNCIONA CORRECTAMENTE:

- EL RANGO ES DESDE 0 HASTA 7, OCHO BLOQUES EN TOTAL
    - COINCIDE CON EL VALOR OBTENIDO EN LA PARTE FINAL: Blockchain on Node A is currently 7 blocks long New Block Received; checking validity... Blockchain on Node A is now 8 blocks long
    - PERO SI SIGO EJECUTANDO EL CODIGO, SE GENERA UN NUEVO BLOQUE...
  - SI EJECUTO LA ORDEN: chain[8], DA ESTE RESULTADO:

## IndexError Traceback (most recent call last)

in () ----> 1 chain[8]

**IndexError: list index out of range**

In [53]: chain[2]

```
IndexError Traceback (most recent call last)
<ipython-input-53-e944d31b458c> in <module>()
      1 chain[2]
```

`IndexError: list index out of range`

```
In [64]: chain[3]
```

```
IndexError                                Traceback (most recent call last)
<ipython-input-64-3c6b96af144a> in <module>()
      1 chain[3]
```

IndexError: list index out of range

COMO SE ESPERABA, EL BLOQUE GENESIS INCLUYE UNA TRANSACCION INVALIDA QUE INICIA LOS BALANCES DE CUENTAS (CREA TOKENS DE LA NADA). EL HASH DE LOS BLOQUES PRECEDENTES SE REFLEJA EN EL BLOQUE HIJO, QUE CONTIENE UN CONJUNTO DE NUEVAS TRANSACCIONES QUE AFECTAN AL ESTADO DEL SISTEMA. PODEMOS VER AHORA EL ESTADO DEL SISTEMA, ACTUALIZADO PARA INCLUIR LAS TRANSACCIONES.

```
In [54]: state
```

## **COMPROBANDO LA VALIDEZ DE LA CADENA**

AHORA QUE CONOCEMOS COMO CREAR NUEVOS BLOQUES Y COMO ENLAZARLOS ENTRE SI FORMANDO UNA CADENA, DEFINAMOS FUNCIONES PARA COMPROBAR QUE LOS NUEVOS BLOQUES SON VALIDOS Y QUE EL TOTAL DE LA CADENA TAMBIEN ES VALIDO.

EN UNA RED BLOCKCHAIN ESTO SE CONVIERTER EN ALGO IMPORTANTE, EN DOS ASPECTOS:

- CUANDO INICIALMENTE PONEMOS EN MARCHA NUESTRO NODO, DESCARGAREMOS LA HISTORIA COMPLETA DEL BLOCKCHAIN. DESPUES DE DESCARGAR LA CADENA, PODEMOS NECESITAR EJECUTAR TODO EL BLOCKCHAIN PARA CALCULAR EL ESTADO DEL SISTEMA. PARA PROTEGERNOS DE QUE ALGUIEN INSERTE TRANSACCIONES INVALIDAS EN LA CADENA INICIAL, NECESITAMOS CHEQUEAR LA VALIDEZ DE TODA LA CADENA TRAS SU DESCARGA INICIAL.
- UNA VEZ QUE NUESTRO NODO ESTÁ SINCRONIZADO CON LA RED (YIENE UNA COPIA ACTUALIZADA DEL BLOCKCHAIN Y UNA REPRESENTACION DEL ESTADO DEL SISTEMA) NECESITARÁ CHEQUEAR LA VALIDEZ DE LOS NUEVOS BLOQUES QUE SE DIFUNDIRAN A LA RED.

NECESITAREMOS 3 FUNCIONES PARA FACILITAR TODO ESTO:

- **checkBlockHash**: UNA FUNCION DE AYUDA SIMPLE QUE ASEGURA QUE EL CONTENIDO COINCIDE CON EL HASH
- **checkBlockValidity**: COMPRUEBA LA VALIDEZ DE UN BLOQUE, DANDO SU PREDECESOR Y EL ESTADO ACTUAL DEL SISTEMA. QUEREMOS QUE DEVUELVA EL ESTADO ACTUALIZADO SI EL BLOQUE ES VALIDO Y QUE GENERE UN ERROR EN CASO CONTRARIO.
- **checkChain**: COMPRUEBA LA VALIDEZ DE TODA LA CADENA Y CALCULA EL ESTADO INICIAL DEL SISTEMA EN EL BLOQUE GENESIS. ESTO DEVOLVERA EL ESTADO DEL SISTEMA , SI ES VALIDO Y QUE GENERA UN ERROR EN CASO CONTRARIO.

```
In [41]: def checkBlockHash(block):
    # Raise an exception if the hash does not match the block contents
    expectedHash = hashMe( block['contents'] )
    if block['hash']!=expectedHash:
        raise Exception('Hash does not match contents of block %s'%block['contents']['blockNumber'])
    return
```

COMENTARIOS SOBRE LA FUNCION: checkBlockHash(block)

- RECIBE UN BLOQUE Y LO GUARDA EN LA VARIABLE: block
- COMPARA EL HASH QUE FIGURA EN EL BLOQUE CON EL HASH DEL BLOQUE QUE ELLA MISMA CALCULA
- SI NO COINCIDEN, LANZA UNA EXCEPCION:

'Hash does not match contents of block %s'%

block['contents']['blockNumber']

```
In [42]: def checkBlockValidity(block,parent,state):
    # We want to check the following conditions:
    # - Each of the transactions are valid updates to the system state
    # - Block hash is valid for the block contents
    # - Block number increments the parent block number by 1
    # - Accurately references the parent block's hash
    parentNumber = parent['contents']['blockNumber']
    parentHash   = parent['hash']
    blockNumber  = block['contents']['blockNumber']

    # Check transaction validity; throw an error if an invalid transaction was found.
    for txn in block['contents']['txns']:
        if isValidTxn(txn,state):
            state = updateState(txn,state)
        else:
            raise Exception('Invalid transaction in block %s: %s'%(blockNumber,txn))

    checkBlockHash(block) # Check hash integrity; raises error if inaccurate

    if blockNumber!=(parentNumber+1):
        raise Exception('Hash does not match contents of block %s'%blockNumber)

    if block['contents']['parentHash'] != parentHash:
        raise Exception('Parent hash not accurate at block %s'%blockNumber)

    return state
```

COMENTARIOS checkBlockValidity(block,parent,state):

- QUEREMOS COMPROBAR LAS SIGUIENTES CONDICIONES:
  - CADA TRANSACCION ES UNA ACTUALIZACION VALIDA AL ESTADO DEL SISTEMA

- EL HASH DEL BLOQUE ES VALIDO PARA EL CONTENIDO DEL BLOQUE
- EL NUMERO DE BLOQUE INCREMENTA EL NUMERO DEL BLOQUE PRECEDENTE (PARENT) EN UNO
- REFERENCIA EXACTAMENTE EL HASH DEL BLOQUE PRECEDENTE
- COMPRUEBA LA VALIDEZ DE LA TRANSACCION, LANZA UN ERROR SI ENCUENTRA UNA TRANSACCION INVALIDA
- COMPRUEBA LA INTEGRIDAD DEL HASH; LANZA UN ERROR SI ES INCORRECTO

COMENTARIOS SOBRE LA FUNCION: checkBlockValidity(block,parent,state)

- RECIBE UN BLOQUE, SU PARENT Y EL ESTADO
- CALCULA Y GUARDA LAS VARIABLES: parentNumber, parentHash, blockNumber
- COMPRUEBA LA VALIDEZ DE UNA TRANSACCION Y LANZA UNA EXCEPCION SI ES INVALIDA
- TAMBIEN COMPRUEBA LA INTEGRIDAD DEL HASH
- SI NO COINCIDEN, LANZA UNA EXCEPCION: Exception('Invalid transaction in block %s: %s' %(blockNumber,tx))

```
In [121]: def checkChain(chain):
    # Work through the chain from the genesis block (which gets special treatment),
    # checking that all transactions are internally valid,
    # that the transactions do not cause an overdraft,
    # and that the blocks are linked by their hashes.
    # This returns the state as a dictionary of accounts and balances,
    # or returns False if an error was detected

    ## Data input processing: Make sure that our chain is a list of dicts
    if type(chain)==str:
        try:
            chain = json.loads(chain)
            assert( type(chain)==list)
        except: # This is a catch-all, admittedly crude
            return False
    elif type(chain)!=list:
        return False

    state = {}
    ## Prime the pump by checking the genesis block
    # We want to check the following conditions:
    # - Each of the transactions are valid updates to the system state
    # - Block hash is valid for the block contents

    for txn in chain[0]['contents']['txns']:
        state = updateState(txn,state)
    checkBlockHash(chain[0])
    parent = chain[0]

    ## Checking subsequent blocks: These additionally need to check
    # - the reference to the parent block's hash
    # - the validity of the block number
    for block in chain[1:]:
        state = checkBlockValidity(block,parent,state)
        parent = block

    return state
```

COMENTARIOS checkChain(chain):

- RECORRE TODA LA CADEN DESDE EL BLOQUE GENESIS (QUE NECESITA DE UN TRATAMIENTO ESPECIAL),
  - CHEQUEA QUE TODAS LAS TRANSACCIONES SEAN VÁLIDAS INTERNAMENTE
    - QUE LAS TRANSACCIONES NO PROVOQUEN UN DESCUBIERTO
    - Y QUE LOS RESPECTIVOS BLOQUES ESTAN ENLAZADOS POR SUS HASHES
      - ESTO DEVUELVE UN ESTADO COMO UN DICCIONARIO DE CUENTAS Y BALANCES,
  - O DEVUELVE FALSE SI SE DETECTA CUALQUIER ERROR
    - PROCESO DE ENTRADAS DE DATOS: SE ASEGURA DE QUE NUESTRA CADENA ES UNA LISTA DE DICTADOS (dicts)
      - ESTO ES UN CAPTURA-TODO, VERDADERAMENTE CRUDO
- PREPARA EL PROCESO CHEQUEANDO EL BLOQUE GENESIS
- QUEREMOS CHEQUEAR LAS SIGUIENTES CONDICIONES:
  - CADA UNA DE LAS TRANSACCIONES SON ACTUALIZACIONES VALIDAS AL ESTADO DEL SISTEMA

- EL HASH DEL BLOQUE ES VALIDO PARA EL CONTENIDO DEL BLOQUE

CHEQUE LOS BLOQUES SIGUIENTES: PARA ELLO ES NECESARIO CHEQUEAR ADEMÁS:

- \* LA REFERENCIA AL HASH DE LOS BLOQUES QU LE PRECEDEN (PARENTS)
- \* LA VALIDEZ DEL Nº DE BLOQUE

COMENTARIOS SOBRE LA FUNCION: checkChain(chain): RECIBE UN BLOQUE, SU PARENT Y EL ESTADO

YA PODEMOS CHEQUEAR LA VALIDEZ DEL ESTADO

In [122]: `checkChain(chain)`

```
-----
TypeError                                     Traceback (most recent call last)
<ipython-input-122-0bf2183a877d> in <module>()
----> 1 checkChain(chain)

<ipython-input-121-c17d9e5175ae> in checkChain(chain)
    33     #      - the validity of the block number
    34     for block in chain[1:]:
----> 35         state = checkBlockValidity(block, parent, state)
    36         parent = block
    37

<ipython-input-42-6bcc2b9300e5> in checkBlockValidity(block, parent, state)
    11     # Check transaction validity; throw an error if an invalid transaction was found.
    12     for txn in block['contents']['txns']:
----> 13         if isValidTxn(txn, state):
    14             state = updateState(txn, state)
    15         else:

TypeError: 'bool' object is not callable
```

And even if we are loading the chain from a text file, e.g. from backup or loading it for the first time, we can check the integrity of the chain and create the current state:

In [123]: `chainAsText = json.dumps(chain, sort_keys=True)`  
`checkChain(chainAsText)`

```
-----
TypeError                                     Traceback (most recent call last)
<ipython-input-123-2acc6b96aa22> in <module>()
      1 chainAsText = json.dumps(chain, sort_keys=True)
----> 2 checkChain(chainAsText)

<ipython-input-121-c17d9e5175ae> in checkChain(chain)
    33     #      - the validity of the block number
    34     for block in chain[1:]:
----> 35         state = checkBlockValidity(block, parent, state)
    36         parent = block
    37

<ipython-input-42-6bcc2b9300e5> in checkBlockValidity(block, parent, state)
    11     # Check transaction validity; throw an error if an invalid transaction was found.
    12     for txn in block['contents']['txns']:
----> 13         if isValidTxn(txn, state):
    14             state = updateState(txn, state)
    15         else:

TypeError: 'bool' object is not callable
```

## JUNTANDOLO TODO: LA ARQUITECTURA BLOCKCHAIN FINAL

(Putting it together: The final Blockchain Architecture)

EN UNA RED BLOCKCHAIN ACTUAL, LOS NUEVOS NODOS DEBEN DESCARGAR UNA COPIA DEL BLOCKCHAIN Y VERIFICARLA (COMO HEMOS HECHO MAS ARRIBA) Y ENTONCES, ANUNCIAR SU PRESENCIA EN LA RED PEER-TO-PEER Y EMPEZAR A ESCUCHAR LAS TRANSACCIONES. AGRUPANDO LAS TRANSACCIONES EN UN BLOCK Y

PASAR SU PROPUESTA DE BLOQUE A LOS OTROS NODOS.

HEMOS VISTO COMO VERIFICAR UNA COPIA DEL BLOCKCHAIN Y COMO AGRUPAR LAS TRANSACCIONES EN UN BLOQUE. SI RECIBIMOS UN BLOQUE DESDE ALGUN OTRO SITIO, VERIFICARLO Y AÑADIRLO A NUESTRO BLOCKCHAIN ES FACIL.

DIGAMOS QUE EL SIGUIENTE CODIGO SE EJECUTA EN EL NODO A, QUE ES EL QUE MINA EL BLOQUE:

```
In [55]: import copy
nodeBchain = copy.copy(chain)
nodeBtxns  = [makeTransaction() for i in range(5)]
newBlock   = makeBlock(nodeBtxns,nodeBchain)

-----
NameError                                 Traceback (most recent call last)
<ipython-input-55-472ed3cc9c09> in <module>()
      1 import copy
      2 nodeBchain = copy.copy(chain)
----> 3 nodeBtxns  = [makeTransaction() for i in range(5)]
      4 newBlock   = makeBlock(nodeBtxns,nodeBchain)

NameError: name 'makeTransaction' is not defined
```

ASUMAMOS AHORA QUE EL NUEVO BLOQUE SE TRANSMITE A NUESTRO NODO Y QUE QUEREMOS CHEQUEARLO Y ACTUALIZAR NUESTRO ESTADO, SI ES QUE ES UN BLOQUE VALIDO:

```
In [56]: # PRUEBA DE IMPRESION DE CONTENIDO DE LAS VARIABLES ANTERIORES

# print(nodeBchain)

# print(nodeBtxns)

print(newBlock)

# FUNCIONAN LAS TRES VARIABLES Y DAN DIFERENTES CONTENIDOS
# PENDIENTE DE COMENTAR CADA UNA DE ELLAS

-----
NameError                                 Traceback (most recent call last)
<ipython-input-56-afb7596da514> in <module>()
      5 # print(nodeBtxns)
      6
----> 7 print(newBlock)
      8
      9 # FUNCIONAN LAS TRES VARIABLES Y DAN DIFERENTES CONTENIDOS

NameError: name 'newBlock' is not defined
```

```
In [57]: print("Blockchain on Node A is currently %s blocks long"%len(chain))

try:
    print("New Block Received; checking validity...")
    state = checkBlockValidity(newBlock,chain[-1],state) # Update the state- this will throw an error if
    chain.append(newBlock)
except:
    print("Invalid block; ignoring and waiting for the next block...")

print("Blockchain on Node A is now %s blocks long"%len(chain))

Blockchain on Node A is currently 2 blocks long
New Block Received; checking validity...
Invalid block; ignoring and waiting for the next block...
Blockchain on Node A is now 2 blocks long
```

COMENTARIO:

- ACTUALIZA EL ESTADO - LANZARA UN ERROR SI EL BLOQUE ES INVALIDO
- LO PRIMERO QUE HACE ES IMPRIMIR EL NUMERO DE BLOQUES QUE TIENE EL BLOCKCHAIN EN EL NODO A
- DESPUES INFORMA DE QUE SE HA RECIBIDO UN NUEVO BLOQUE Y QUE ESTA CHEQUEANDO SU VALIDEZ

- LANZARA UN ERROR SI EL BLOQUE ES INVALIDO
  - SI LA ACTUALIZACION ES CORRECTA IMPRIME LA NUEVA LONGITUD DE LOS BLOQUES DEL BLOCKCHAIN EN EL NODO A

```
In [58]: # PRUEBA DE IMPRESION DE CONTENIDO DE LAS VARIABLES ANTERIORES
```

```
print(chain)
```

```
In [59]: print(state)
```

## **CONCLUSIONES Y EXTENSIONES:**

HEMOS CREADO TODA LA ARQUITECTURA BASICA PARA UN BLOCKCHAIN, DESDE UN CONJUNTO DE REGLAS DE TRANSICIONES ENTRE ESTADOS HASTA UN METODO PARA CREAR BLOQUES, TAMBIEN MECANISMOS PARA CHEQUEAR LA VALIDEZ DE LAS TRANSACCIONES, DE LOS BLOQUES Y DE LA CADENA COMPLETA. PODEMOS DERIVAR EL ESTADO DEL SISTEMA DESDE UNA COPIA DESCARGADA DEL BLOCKCHAIN, VALIDAR LOS NUEVOS BLOQUES QUE RECIBAMOS DESDE LA RED, ASI COMO CREAR NUESTROS PROPIOS BLOQUES.

EL ESTADO DEL SISTEMA QUE HEMOS CREADO ES EFECTIVAMENTE UN LEDGER (LIBRO MAYOR) DISTRIBUIDO O BASE DE DATOS Y ES EL CORAZON DE MUCHOS BLOCKCHAINS. PODEMOS AMPLIARLO PARA INCLUIR TIPOS DE TRANSACCIONES ESPECIALES O CONTRATOS INTELIGENTES COMPLETOS.

NO HEMOS EXPLORADO LA ARQUITECTURA DE LA RED, LOS PASOS DE VALIDACION DE LAS PRUEBAS DE TRABAJO O DE LAS PRUEBAS DE ESTADO NI EL MECANISMO DE CONSENSO QUE PROPORCIONA SEGURIDAD A LOS BLOCKCHAINS FRENTE A ATAQUES. TAMPOCO HEMOS DISCUSIDO SOBRE LA CRIPTOGRAFÍA DE CLAVE PUBLICA, PRIVACIDAD Y PASOS DE VERIFICACION. ¡SERA EN UN FUTURO!

---

---

---

---

