Access Coin instructions

1. You will need to import the following modules:

```
from primes import * # to sign transactions
from coins import * # hashes, transactions, blocks etc...
import coins_client  # to talk to the server
from pubkeys import * # the public keys from Thursday
```

2. Define your private key by constructing an object:

```
privkey = PrivateKey(N=943,
         e=7,
         d=504)
```

3. The module `pubkeys` imports a variable called `pubkeys` that contains the public keys from all groups as a list of 4 elements (as objects of class PublicKey) and the instructor public key (as an object of class PublicKey). So `pubkeys[i % 4]` is the public key of group i (assuming there are 4 groups). In Python, `i % \ngroups` calculates $i \mod 4$. If `i` is your group number you can check that the public part of your private key matches the ones in the module `pubkeys`

```
privkey.pubkey() == pubkeys[i % \ngroups]
```

4. Bitcoin and other cryptocurrencies are peer to peer, meaning many nodes in the network have a copy of the blockchain. Keeping track of the blockchain involves tedious book-keeping and error checking, so we will just assume that one server has the blockchain. The URL for the server will be provided, it will be something like `http://someserver.utah.edu:8080/`. We point our code to the right URL:

```
coins_client.base_url = "http://someserver.utah.edu:8080/"
```

5. It's time to mine some coins!

   (a) First ask the server for a new block, specifying the public key `pubkey` to whom the reward for mining this block will be assigned.

```
b = coins_client.get_newblock(pubkey)
```

(b) Doing `b.check_nonce()` returns `True` if the block nonce is OK or `False` otherwise.

(c) Mining the block `b` means to try many different values for the integer `b.nonce`, until `b.check_nonce()` is `True`.

(d) You can do this manually by incrementing the nonce until the nonce is good, but it is much better to write a `while` loop.

(e) Once we have a good nonce we can submit the block to the blockchain server:

```
status,msg = coins_client.post_newblock(b)
```

If the block was accepted by the server, `status` is `True`. Otherwise, `msg` contains a message describing the error.

(f) Congratulations! You now have 10 Access Coins.

(g) Repeat (a) – (f) another time, so that you have 20 Access Coins.
Please do not mine more than two blocks and wait until all the groups have mined 2 blocks before continuing with the next step!

(h) Think of other creative ways of mining. Maybe the mining itself can be put in a loop? Maybe you can have several computers/processes mining for you?

6. **Signing.** Complete the function `txsig` that signs a `Transaction` `tx` with the `PrivateKey` `privkey`. The order of operations is as follows.

(a) Compute the hash of the transaction, `tx.hash()` will do the trick and give you a string with exactly $n = 32$ characters.

(b) Encode the hash using the Davis table into an integer with $2n = 64$ digits (use `primes.davis_enc(h)`, where `h` is the hash string)

(c) Decrypt this integer with the private key that is supplied as an argument

(d) return the result

7. **Transactions**. Let's say that Group A wants to buy an espresso from Group B, for x coins. Let `pubkeyA` and `pubkeyB` be their public keys and let `privkeyA` be the private key of Group A.

(a) Group A asks the server to list all the transactions that are in favor of Group A (and that have not been spent yet)

```
wallet = coins_client.get_wallet(pubkeyA)
```

(b) Group A creates a transaction from using one or more of these unspent transactions that appear in the wallet . The function `tx_send_funds` does this and makes sure there is no overspending!

```
tx = coins_client.tx_send_funds(wallet,
                                pubkeyA,pubkeyB,
                                x,"For a frothy espresso")
```

(c) Group A uses `privkeyA` and the function `tx_sig` to sign the new transaction `tx`. The signature is an integer, say `sig`. Anyone can check that Group A wants this transaction as is, since encrypting `sig` with the `pubkeyA` should give `tx.hash()` (in Davis coding). Any attempt to tamper with the transaction would change the hash and the transaction signature will not check out anymore!

(d) Group A posts the transaction to the server.

```
status,msg = coins_client.post_tx(tx,sig)
```

(e) Group C (which may be A, B, or a different group) will eventually get a block with this new transaction included in it, mine it and submit to the server. Once Group B gets confirmation from the server that the transaction has cleared, Group B can give Group A an espresso!

Note: Please no eating or drinking in the computer lab!