

TD2 Blockchain Programming

Samuel Barbarin – Nicolas Dumont

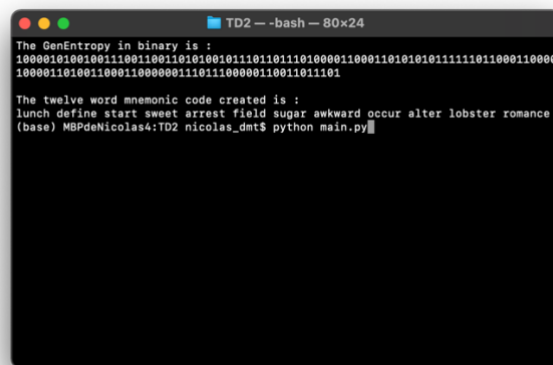
BIP 39 :

- Créer un repo GitHub et le partager avec le prof :

<https://github.com/nicolasdmt/TD2>

- Créer un programme python ou JS interactif en ligne de commande :

Notre programme main peut être exécuté dans la console à l'aide de la commande : `python main.py`.



```
TD2 -- -bash -- 80x24
The GenEntropy in binary is :
1000010100100111001100110101001011101101101000011000110101011111011000110000
100001101001100011000000111011100000110011011101
The twelve word mnemonic code created is :
lunch define start sweet arrest field sugar awkward occur alter lobster romance
(base) MSPdeNicolas4:TD2 nicolas_dmt$ python main.py
```

- Créer un entier aléatoire pouvant servir de seed à un wallet de façon sécurisée

```
seed = secrets.randbits(128)
```

A l'aide de la bibliothèque `secrets` on génère un entier aléatoire de 128 bits de façon sécurisée.

- Représenter cette seed en binaire et la découper en lot de 11 bit

```
genEntropy = bin(seed)[2:]
```

Cette commande permet d'obtenir le nombre aléatoire en binaire. On va par la suite le hasher avec la bibliothèque `hashlib` et sa méthode `sha256` de manière à pouvoir faire le checksum et obtenir l'entropy. On peut enfin diviser l'entropy en 12 lots de 11 bits et les associer à des mots avec la wordlist BIP 39 grâce à la méthode ci-dessous.

```
def creationMnemonic (entropy):
    with open("wordList.txt", 'r') as list:
        wordList = [w.strip() for w in list.readlines()]

    mnemonic = ""
    for i in range(0, 12, 1):
        index = int(entropy[11 * i:11 * (i + 1)], 2)
        mnemonic = mnemonic + wordList[index] + " "
    return mnemonic
```

- Attribuer à chaque lot un mot selon la liste BIP 39 et afficher la seed en mnémonique

Voir étape précédente et la méthode associée. Pour afficher la seed cela se passe dans le main avec :

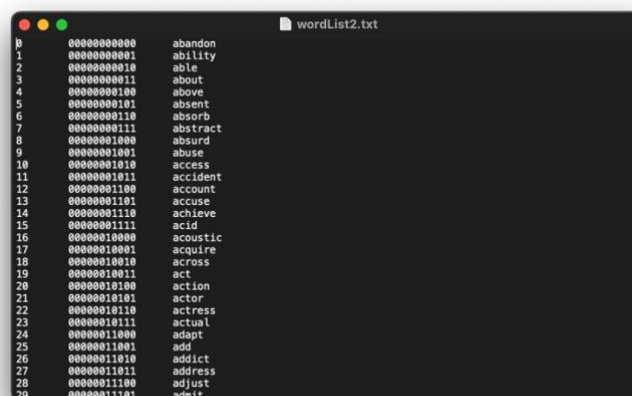
```
mnemonic = creationMnemonic(entropy)
print("\nThe twelve word mnemonic code created is : \n" + mnemonic)
```

Lorsqu'on essaye de vérifier les mots que l'on obtient sur le site qui a été fourni, <https://iancoleman.io/bip39/>, ils ne sont pas valides il y a donc un problème dans notre code. Cela vient sans doute d'un problème qui avait été abordé en TD, la perte d'informations possible à la suite du passage d'un objet en string ou en bin. Une fonction de la bibliothèque hashlib, encode(), que l'on a dû utiliser pour faire un sha 256 nous intrigue et nous pensons que le problème vient de là mais nous n'avons pas su y remédier.

```
seedEnc = genEntropy.encode()
seedHash = hashlib.sha256(seedEnc)
```

- Permettre l'import d'une seed mnémonique

Pour cette étape, nous n'arrivons pas à associer les mots aux lots de bits correspondants. Nous avons trouvé un fichier txt qui associe les mots aux lots mais nous n'arrivons pas à l'exploiter.



Il est sous cette forme mais faute de temps nous n'arrivons pas à le modifier pour retirer les premiers caractères de chaque ligne ou alors accéder seulement aux caractères qui nous intéressent, les lots de bits et les mots, comme les espaces ne sont pas les mêmes à chaque ligne.

```
for i in range(0, 12, 1):
    for j in range(0, 2048):
        if j < 10:
            if mnemonic[i] == wordList[j][12:]:
                entropy += wordList[j][1:13]
        elif j < 100:
            if mnemonic[i] == wordList[j][13:]:
                entropy += wordList[j][2:14]
        elif j < 1000:
            if mnemonic[i] == wordList[j][14:]:
                entropy += wordList[j][3:15]
        else:
            if mnemonic[i] == wordList[j][15:]:
                entropy += wordList[j][4:16]
```

BIP 32 :

Dans cette partie, nous avons utilisé des bibliothèques comme Mnemonic et bip32utils.

```
from mnemonic import Mnemonic
import bip32utils
```

- Extraire la master private key et le chain code

```
root_key = bip32utils.BIP32Key.fromEntropy(seed)
chain_code = root_key.ChainCode().hex()
root_private_hex = root_key.PrivateKey().hex()
root_public_hex = root_key.PublicKey().hex()
```

- Extraire la master public key

```
root_public_hex = root_key.PublicKey().hex()
```

- Générer une clé enfant

```
child_key = root_key.ChildKey(0)
child_public_hex = child_key.PublicKey().hex()
child_private_hex = child_key.PrivateKey().hex()
```

Il s'agit ici de la première clé enfant.