In [1]:
```python
import hashlib
import time
import re
class VPN_Proxy:
    def __init__(self):
        self.mapper = "abcdef0123456789"
        self.Database = {}

    def encode(self, ip: str) -> str:
        ipType = self.validIPAddress(ip)
        if ipType == 'Neither':
            raise Exception("NOT VALID IP")
        tagID = ""
        hash = hashlib.sha512()
        while tagID in self.Database.keys() or tagID == "":
            tagID = ""
            hash.update(str(time.time()).encode('utf-8'))
            string = hash.hexdigest()
            for i in range(0, len(string), 21):
                tagID += self.mapper[self.decodeHex2Int(string[i:i+33])%16]
        self.Database[tagID] = ip
        return tagID

    def decode(self, tagID: str) -> str:
        if tagID in self.Database.keys():
            return self.Database[tagID]
        return ""

    def signingOff(self, tagID):
        if tagID in self.Database.keys():
            del self.Database[tagID]

    def decodeHex2Int(self, hexcode):
        dictionary = {'a':10, 'b':11, 'c':12, 'd':13, 'e':14, 'f':15}
        ints = 0
        for x in range(0, len(hexcode)):
            i = hexcode[x]
            if i != ":":
                first = i.lower()
                if first in dictionary.keys():
                    first = dictionary[first]
                ints += int(first)
        return ints

    def validIPAddress(self, IP: str) -> str:
        patternIPv4 = "^((([1]?[1-9]?[0-9])|([2][0-4][0-9])|([2][0-5][0-5])|([1][0-9][0-9])).){3}(([1]?
[1-9]?[0-9])|([2][0-4][0-9])|([2][0-5][0-5])|([1][0-9][0-9]))$"
        patternIPv6 = "^([0-9a-fA-F]{1,4}[:]){7}[0-9a-fA-F]{1,4}$"
        if re.search(patternIPv4, IP):
            return "IPv4"
        elif re.search(patternIPv6, IP):
            return "IPv6"
        return "Neither"

    def returnDatabaseLength(self):
        return len(self.Database)
```

In [2]:
```python
proxyAdd = VPN_Proxy()
```

In [3]:
```python
def generateIPv6Addresses(amount):
    import random
    examples = []
    dictionary = {10:'a', 11:'b', 12:'c', 13:'d', 14:'e', 15:'f'}
    for i in range(amount):
        temp = ""
        for x in range(8):
            section = ""
            for y in range(random.randint(1,4)):
                character = random.randint(0,15)
                if character > 9:
                    character = dictionary[character]
                section += str(character)
            temp += section + ":"
        examples.append(temp[:-1])
    return examples
```

In [4]:
```python
def stripLeadingZeros(string):
    while len(string) > 1 and string[0] == '0':
        string = string[1:]
    return string

def generateIPv4Addresses(amount):
    import random
    examples = []
    for i in range(amount):
        temp = ""
        for x in range(4):
            section = ""
            for y in range(random.randint(1,3)):
                if section == "":
                    character = random.randint(0,2)
                elif section[0] == '2' and len(section) == 1:
                    character = random.randint(0,5)
                elif section[0] == '2' and section[1] == '5':
                    character = random.randint(0,5)
                else:
                    character = random.randint(0,9)
                section += str(character)
            temp += stripLeadingZeros(section) + "."
        examples.append(temp[:-1])
    return examples
```

In [5]:
```python
ipv4Tests = generateIPv4Addresses(100)
ipv6Tests = generateIPv6Addresses(100)
```

In [6]:
```python
for i in range(len(ipv4Tests)):
    temp = proxyAdd.encode(ipv4Tests[i])
    ipv4Tests[i] = temp
    back = proxyAdd.decode(temp)
    print(back, temp)
```

```
2.16.225.204 a96a11f
159.0.18.45 d46e187
254.7.5.1 ca885b4
14.36.147.1 08e5349
1.24.243.77 0b93353
2.0.0.250 48cf94f
2.5.1.2 8d88b0a
25.1.13.2 fb30a1e
139.2.22.86 77b9c4c
2.23.254.7 42a9750
44.13.1.23 3a42d54
10.0.1.1 681aad7
1.5.6.1 f76f6d4
84.1.0.203 5f6ac6b
24.2.134.2 a0d6047
19.20.1.193 27995ae
1.101.2.43 e652d6f
253.0.2.1 55e4be5
20.87.1.0 387c5c8
2.247.2.221 c0ba945
23.14.200.10 78e5a35
1.22.140.0 0256510
2.20.15.2 61e010d
18.2.3.17 8695d82
1.23.0.1 73aa4e7
0.13.0.22 ef2352b
234.194.42.18 80aadbb
1.0.2.0 a11be7a
138.23.2.0 dd1a954
22.2.2.23 bbcd040
20.2.10.2 2b96eb2
2.65.11.25 bb85eaf
52.105.212.253 ad18293
7.1.206.85 0d3be75
1.1.42.0 e575a42
73.1.1.10 9825919
178.109.0.12 6469bbd
2.18.2.124 99ce4a6
16.2.2.131 2de1949
20.25.104.15 c03237e
25.18.2.0 bdd3795
1.20.2.18 b4aca8b
15.20.21.2 6313a50
2.0.0.1 5221ef0
1.24.24.140 db79246
18.2.17.159 66abe1f
222.1.21.246 e215894
41.249.1.0 4f79915
99.8.233.0 3e56a8f
0.117.215.246 4dc0797
2.2.139.248 dc0e42f
2.12.2.2 158ebfe
254.2.116.255 bda2534
24.194.235.1 48327bd
2.1.2.170 ce31ed8
8.1.0.2 c36e1af
62.1.17.25 002903f
0.25.1.2 9c8dda4
1.17.0.0 46d9f70
227.1.0.2 c2cc2f1
6.222.22.12 73c4aae
4.164.1.244 1f3d91b
92.2.19.9 612c9c8
0.9.15.239 daf7998
10.22.212.2 d645121
249.169.5.134 a706e54
134.1.1.0 73e69cf
9.0.240.0 9060585
8.1.123.2 fc6e2ae
237.2.41.69 fdab725
4.12.11.16 b81d07e
1.237.15.6 016d72f
240.67.0.251 84afadf
8.0.1.1 62c76ea
56.209.0.1 2cc8122
2.7.2.177 2127a61
2.10.2.1 c5ee52b
10.1.2.1 e658e18
0.2.10.1 0049607
2.230.2.21 11d005e
1.1.1.234 c24b994
99.28.208.249 faf058b
14.21.1.18 51902c2
108.212.0.1 ad59888
9.33.40.20 6082561
242.2.9.1 2c5199a
0.13.2.181 ef402e2
158.1.13.2 d1cc4b1
3.231.2.6 113838d
20.10.219.221 8e00c05
79.165.1.136 78b1d6d
240.115.23.0 614a9a9
2.232.67.81 6166903
2.58.24.142 d68f86b
17.2.23.0 f1957c4
1.200.1.153 a3ae111
151.0.171.8 f90485a
2.2.121.21 4bddb02
1.22.167.23 2acaf75
1.23.14.1 61c72e7
```

In [7]:
```python
for i in range(len(ipv6Tests)):
    temp = proxyAdd.encode(ipv6Tests[i])
    ipv6Tests[i] = temp
    back = proxyAdd.decode(temp)
    print(back, temp)
```

```
de7e:e199:d:8e:be8:5:ef5:0e17 cd75909
da:f47:20:818:4:6:6fe:90f1 a86f490
d9:55b6:e01:bd:fa:48:12:d2ce e3505e7
b75f:d5e1:755:96:53ff:9:4f2:b1 6fd5c71
dd:47:7:5a9:52a:32:11:77 e795b73
e1a:d1:4:c18c:b:4:ff7:1e5 b5f3d33
065:14b:1306:b721:2fe5:efe:4:f059 402d0f5
1d88:f97:74:96:08fd:1:8300:bf2 5ccfaff
5de:b1:6:7a:401:5:59e:480 606eed5
b1c:883:507:32:5f94:770:cdf:4 6640475
1f:dd:9:a:a034:d2b4:e3b:c 5bb0888
e514:82c:5e3c:ffa:3:6:d63:35 9c6ce0a
59e:a:d6:2f1e:f8:f84:c:c 43701fb
8:352:0:e28:11e:47a:14a:5 3a2fd4a
4f:ccda:73:2:7c7:450d:f0b7:ec de73d3f
c02a:dc:73:730:3e:7b:9ba:379 01f8eb9
23d4:48a:7455:8:fe4:8e:40:3 d0b8458
58:5c98:f64:2:4f2:8:cd:f6c8 fa533c5
9:8122:2:4:1a:48:83:4c 92228a4
34:6b:ac0:bdea:1f:d6:887c:34c1 6aa335c
07:f:3c73:96:30:e:c:0a 0886581
c8e2:f40:00a:3a:7b3d:c:189:2d 07bab2f
dc:cec:a4:7:2:78:a:44 4e8c91a
fbd:46c:65:ff:284b:d:da:44c1 174a157
61b6:9976:0:2:4:36:4694:66 094ab40
8f6:1f:b:69e:2a:e84:5ca:8 3c738f8
fd25:8a48:81:4:259f:2:a:16ed 7e54d52
6c:f1c:9:c040:5e2:05f5:6f:7 df68127
75:e:b43b:ada3:8e4:d5f8:612:3f a2a215f
5c:1d80:04:bd53:d3a1:1:9a:4 530ce41
e496:3:d:a5a6:4:a:dd44:a66a 0c0eb6a
f:c719:57ba:1709:e6:616:78:b5 7581823
82:de:ab:ca8:907:01df:e94:00 013ec32
19:4a:4aa:a2bd:9:d383:d66:4 3b7bbb1
3d:590b:e:3:6030:ac:d28d:d12 a3d75b1
0ac:e8:7:f1:81d:6:c:c cf5fc6c
86a:17fb:fe8f:7204:8df2:32b:9e:33 cff9a25
6:69e:a42:14a:b:4b6:a:5d 2803b3e
f155:125a:a3b:cddd:574:7f6:3b67:3a0f 778c5d7
7c:f:8d4:b538:183:aa:c175:f4 42cf667
f:81:c541:45:018b:9:f78:6e72 f202592
9:edf:d2d:27:fd:a:60c6:9 c22d4d2
951:3:14:2dde:e358:ab:f2e:bd8f 6e4b08b
af7:29:c9b6:b13:c:b46f:0a9f:c8d3 6365464
989:00:c652:9:634:c:d7:7f 08dc036
7:0d:c5:3ab0:ba:67a7:288:00c 81d749d
5abb:0277:8:60f:799:a9b:aa:8416 ce7fb2f
4dea:e:5e6b:857:c60:b0c:a9:a7a f3feaf7
ca:3:aa7:3a7:3b:96d:942:00 772c408
3c:55:1e47:0ea:da:13:0:6da 11d7bec
1:e:5:d757:c8:0b26:33:c43 e0d4422
6:0f:a61a:17:a38:7:c4ea:2 14d698a
52:989:4:8b27:8b76:b209:a3:58e b70736c
54e:4ba:ff:5f:36e:f2f:3:da94 643461a
16d4:636f:b0:ac0:59d:f8f9:2e:3a 41eea18
7:5:3:5467:4b:1:09e:69 f148260
67a:dd0:c485:912:cf:27:0:7ea6 ada8e5c
8b:088:5a:b:6:f:caa:24 09501c9
9ef:e:3:15a6:d0:b8f:0:1c c32fa5c
4:73:2:c634:8f:2ac:1:0 64e4931
6:9:0c:44:b5f:1e:1:aa73 328f3c4
32:a:17:169:4f:f8:c70:3f fbe8d27
763:d:9a:bfd4:8f8c:6dc:6f7:a29 afd62d1
95a:e83:dd21:a:57:22e7:7e:a a432af4
121:52ee:023f:0c38:e4a:ba3:fa1:f478 e8b4c57
ff3:ef:b0a:f:c:99:c216:9f3a 93fb160
0e9:c8:8:99:8aba:c60:fc6a:1 8377423
6:123:bf7:179:5b36:f5:a9:26 74bcacd
62:8:696e:f4f:344:bd07:5f:4d 66b403f
f:84:4:8:14:c2c3:a:8 f186527
2:a614:5f72:e1:4:a7a0:fee6:9 716216b
6e:0:e823:12ec:83:6:a:a83 2b62810
470:86fa:f291:b:fd:444:2dab:fb87 3572ba1
3bd:6bb6:4c2:b:ca:c2:8:f7 8e8bada
4c01:c:13d:e747:09:df:e6:cce 8b091c0
7:7c8:9d:3:2dc:35:3:f ff45b6a
528:1:4c:91:da5:f:22:8b 377d4c8
ac1f:00:8ef5:3d:5c56:c4:ab41:c ecd6352
d9bf:d599:6c:0b:4150:fa0:7fc:3a31 b4010c7
77:c:fc:543b:3f:f:07:6f7 1f3e209
568:9:fe:130:4:71:9:b0c9 4004a8b
09c2:00:490:bdbd:9b69:92b:e:a dd43a8c
19:7:5bd5:749:7bf8:2e:4c36:4be 78dcbdd
4:5bb:4ce2:eb:945:87:0f:1 7b0bb4c
f:76d:a56f:e:5:4d:5e8:2b 5992e43
759:821:25:5:52f:8:62:fde0 18e13aa
0c:c5b:17e3:c2f0:4cf:b8a4:d:e7 c8358d8
c51:cd:d1:15:03:4:980:49 7f34cf4
b609:e5b:d61:5d:a4b:c302:e6:9b 4bd413e
ea:e:61c0:28f:e0f:b4:033:f1a fe96099
0a50:a5:65:fe1:a7:fc0:ad9:1f 1afb462
a005:41be:69:fe:e:3b5:ca8b:f2bd f87d0fb
1d:e1:82:1:af2:c0:2:5 c8ae0e0
1612:d:9:a9:fa01:c912:0:8 7a88734
a2bf:416:ae3e:5:42:8f27:0:93c9 2892b4e
9d0:f:7:5b:67:df9:a1:d2 219d1ca
d:3cef:57e:d:8e23:48d:b:a1a 1adf7fd
f789:58e8:981:97:484a:42:7:d4e 06dfd60
fb38:f9f:c388:c:306:6268:81:6f2b fcaea0a
3fee:bef:7:091:faae:5037:f16:cc37 34d423e
```

In [8]:
```python
print(proxyAdd.returnDatabaseLength())
```

```
200
```

In [9]:
```python
for x in ipv4Tests:
    proxyAdd.signingOff(x)
for y in ipv6Tests:
    proxyAdd.signingOff(y)
print(proxyAdd.returnDatabaseLength())
```

```
0
```

In [ ]: