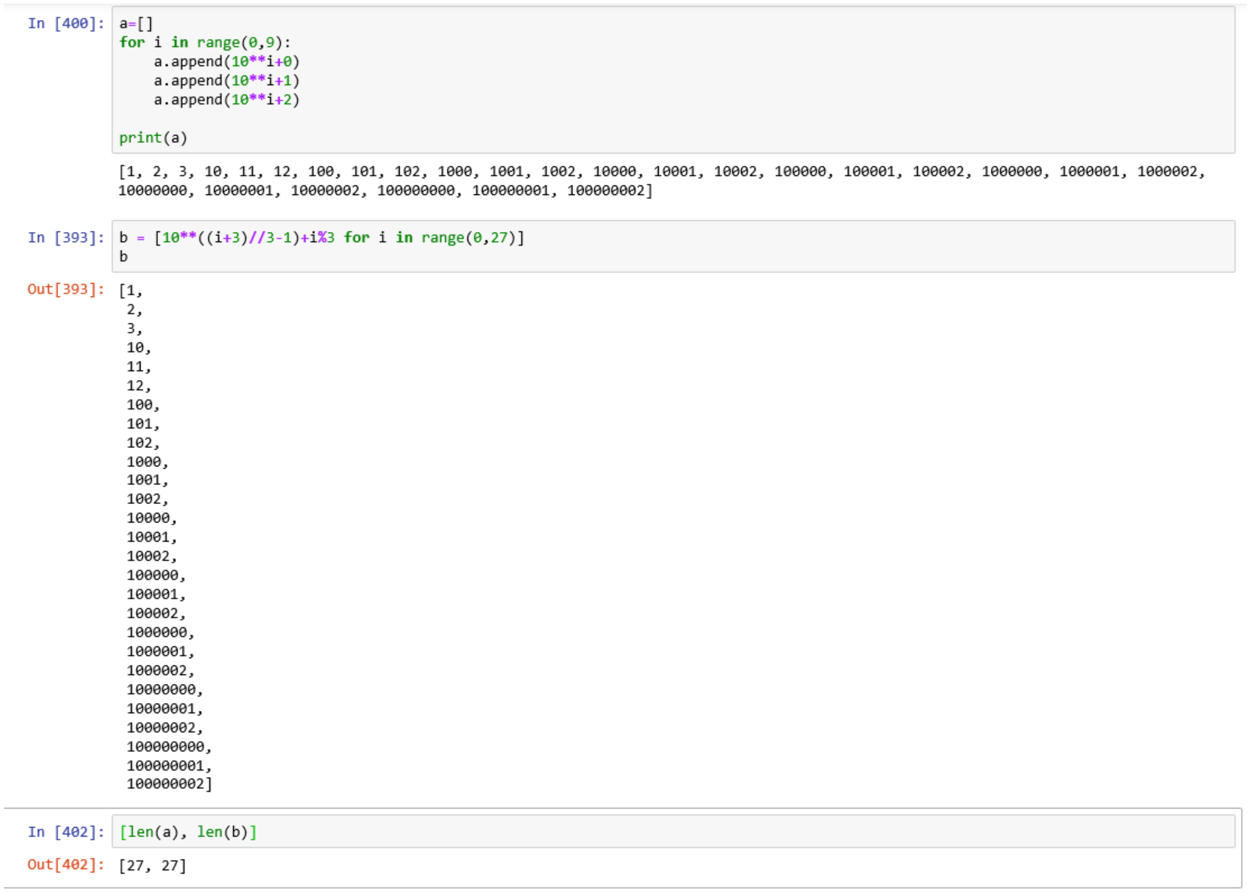
Q 2.1)

List of several numbers have been given and it is asked to regenerate them using for-loop and list comprehension. The following screenshot is the answer for 1.2a to 2.1d

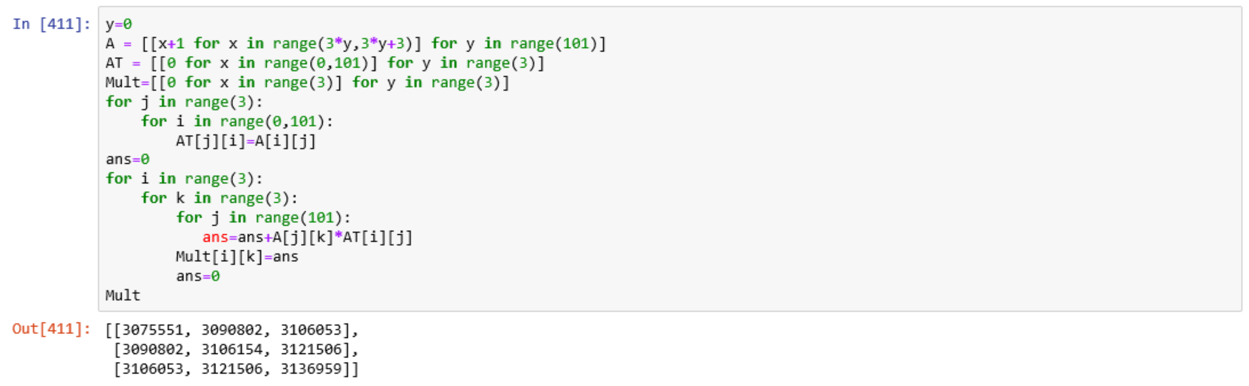


```
In [400]: a=[]
          for i in range(0,9):
              a.append(10**i+0)
              a.append(10**i+1)
              a.append(10**i+2)

          print(a)

          [1, 2, 3, 10, 11, 12, 100, 101, 102, 1000, 1001, 1002, 10000, 10001, 10002, 100000, 100001, 100002, 1000000, 1000001, 1000002,
          10000000, 10000001, 10000002, 100000000, 100000001, 100000002]

In [393]: b = [10**((i+3)//3-1)+i%3 for i in range(0,27)]
          b

Out[393]: [1,
           2,
           3,
           10,
           11,
           12,
           100,
           101,
           102,
           1000,
           1001,
           1002,
           10000,
           10001,
           10002,
           100000,
           100001,
           100002,
           1000000,
           1000001,
           1000002,
           10000000,
           10000001,
           10000002,
           100000000,
           100000001,
           100000002]

In [402]: [len(a), len(b)]

Out[402]: [27, 27]
```

Both methods resulted in identical answers. List comprehensive took me 2 hours!

Q2.2)

In this question, the multiplication of the $A^T A$ is required. At first, A^T was generated and then the multiplication was performed. The following is the code and the results in Python:



```
In [411]: y=0
          A = [[x+1 for x in range(3*y,3*y+3)] for y in range(101)]
          AT = [[0 for x in range(0,101)] for y in range(3)]
          Mult=[[0 for x in range(3)] for y in range(3)]
          for j in range(3):
              for i in range(0,101):
                  AT[j][i]=A[i][j]
          ans=0
          for i in range(3):
              for k in range(3):
                  for j in range(101):
                      ans=ans+A[j][k]*AT[i][j]
                  Mult[i][k]=ans
                  ans=0
          Mult

Out[411]: [[3075551, 3090802, 3106053],
           [3090802, 3106154, 3121506],
           [3106053, 3121506, 3136959]]
```

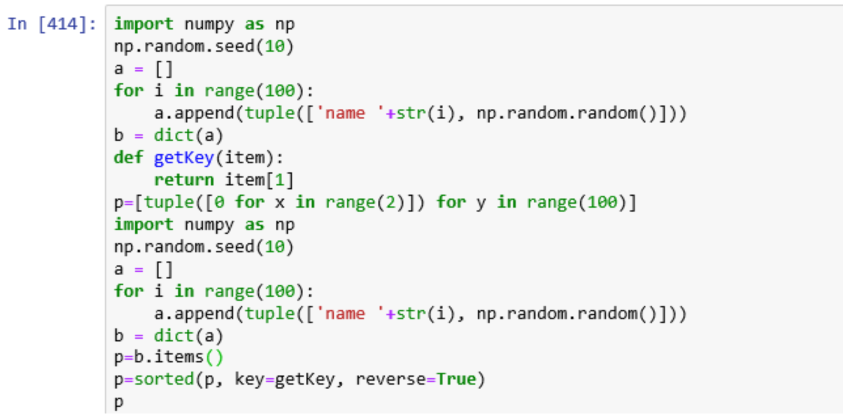
Then results were compared with MMULT function in EXCEL. The EXCEL file is attached:

Q2.3)

The provided list is after ordering in descending order based on value of the dictionary:

```
[('name 56', 0.9876254749018722), ('name 93', 0.9788191457576426), ('name
96', 0.9770807259226818), ('name 11', 0.9533933461949365), ('name 55',
0.9442607122388011), ('name 18', 0.9177741225129434), ('name 32',
0.9086488808086682), ('name 61', 0.9028317603316274), ('name 47',
0.8839364795611863), ('name 76', 0.879915174517916), ('name 78',
0.8780966427248583), ('name 43', 0.8568503024577332), ('name 37',
0.8286813263076767), ('name 58', 0.8261228438427398), ('name 52',
0.8211056578369285), ('name 41', 0.8192869956700687), ('name 14',
0.8126209616521135), ('name 30', 0.8052231968327465), ('name 69',
0.7738302962105958), ('name 0', 0.771320643266746), ('name 77',
0.763240587143681), ('name 7', 0.7605307121989587), ('name 45',
0.7546476915298572), ('name 3', 0.7488038825386119), ('name 16',
0.7217553174317995), ('name 19', 0.7145757833976906), ('name 10',
0.6853598183677972), ('name 23', 0.6741336150663453), ('name 28',
0.6503971819314672), ('name 73', 0.6364911430675446), ('name 2',
0.6336482349262754), ('name 39', 0.6262871483113925), ('name 26',
0.6177669784693172), ('name 15', 0.6125260668293881), ('name 80',
0.6055775643937568), ('name 29', 0.6010389534045444), ('name 82',
0.5978366479629736), ('name 60', 0.5973716482308843), ('name 63',
0.5902013629854229), ('name 89', 0.5655070198881675), ('name 88',
0.5575781886626442), ('name 40', 0.5475861559192435), ('name 20',
0.5425443680112613), ('name 62', 0.5345579488018151), ('name 31',
0.5216471523936341), ('name 99', 0.5197969858753801), ('name 81',
0.5134666274082884), ('name 27', 0.5131382425543909), ('name 13',
0.5121922633857766), ('name 4', 0.4985070123025904), ('name 95',
0.4950486308824543), ('name 90', 0.47513224741505056), ('name 57',
0.4563045470947841), ('name 24', 0.4418331744229961), ('name 97',
0.4407738249006665), ('name 25', 0.4340139933332937), ('name 71',
0.42949217843163834), ('name 79', 0.41750914383926696), ('name 50',
0.3925292439465873), ('name 54', 0.3841144486921996), ('name 22',
0.3733407600514692), ('name 65', 0.3571817586345363), ('name 44',
0.3516526394320879), ('name 74', 0.34634715008003303), ('name 94',
0.33970784363786366), ('name 68', 0.330719311982132), ('name 48',
0.3255116378322488), ('name 33', 0.3192360889885453), ('name 98',
0.3182728054789512), ('name 72', 0.3149268718426883), ('name 67',
0.30545991834281827), ('name 86', 0.30306256065103476), ('name 84',
0.30087130894070724), ('name 35', 0.30070005663620336), ('name 46',
0.2959617068796787), ('name 91', 0.2927979762895091), ('name 17',
0.29187606817063316), ('name 83', 0.2622156611319503), ('name 59',
0.25137413420705934), ('name 87', 0.24207587540352737), ('name 5',
0.22479664553084766), ('name 42', 0.1989475396788123), ('name 6',
0.19806286475962398), ('name 8', 0.16911083656253545), ('name 49',
0.16501589771914849), ('name 53', 0.15115201964256386), ('name 21',
0.14217004760152696), ('name 36', 0.11398436186354977), ('name 51',
0.0934603745586503), ('name 34', 0.09045934927090737), ('name 9',
0.08833981417401027), ('name 66', 0.07961309015596418), ('name 92',
0.06425106069482445), ('name 38', 0.04689631938924976), ('name 75',
0.04309735620499444), ('name 70', 0.039959208689977266), ('name 64',
0.03928176722538734), ('name 85', 0.025399782050106068), ('name 1',
0.020751949359401 5), ('name 12', 0.003948266327914451)]
```

And the written code is as follows:



```
In [414]: import numpy as np
np.random.seed(10)
a = []
for i in range(100):
    a.append(tuple(['name '+str(i), np.random.random()]))
b = dict(a)
def getKey(item):
    return item[1]
p=[tuple([0 for x in range(2)]) for y in range(100)]
import numpy as np
np.random.seed(10)
a = []
for i in range(100):
    a.append(tuple(['name '+str(i), np.random.random()]))
b = dict(a)
p=b.items()
p=sorted(p, key=getKey, reverse=True)
p
```

The highest value's key is: `'name 56'` and the number is `0.98762547490187 22`

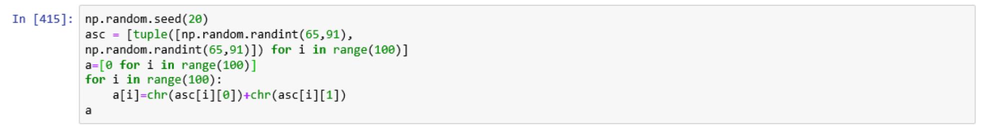
The lowest value and its key are: `'name 12', 0.003948266327914451`

Q2.4)

The provided character list has been regenerated as follows:

```
['DP', 'JU', 'LW', 'HC', 'AW', 'IV', 'TQ', 'GG', 'QJ', 'FH', 'FC', 'GN',
'LZ', 'DK', 'LN', 'TZ', 'OA', 'KL', 'GL', 'SD', 'NR', 'YL', 'QS', 'PH',
'LW', 'NG', 'OY', 'SB', 'OK', 'DS', 'GJ', 'UW', 'OE', 'LI', 'GP', 'SO',
'TK', 'KB', 'VS', 'RI', 'SK', 'UE', 'IW', 'QT', 'UQ', 'VK', 'GJ', 'XU',
'AM', 'OW', 'RJ', 'GK', 'HJ', 'XW', 'JO', 'YY', 'PS', 'MK', 'SR', 'MF',
'CZ', 'XF', 'VZ', 'WJ', 'JV', 'CP', 'ZW', 'BY', 'RQ', 'IL', 'FA', 'WF',
'EL', 'SJ', 'ZO', 'UF', 'DL', 'JX', 'BC', 'TW', 'NV', 'UU', 'RN', 'YL',
'XB', 'JS', 'TE', 'UM', 'EJ', 'KA', 'PV', 'PK', 'ED', 'MK', 'CH', 'AL',
'WP', 'SH', 'LV', 'HX']
```

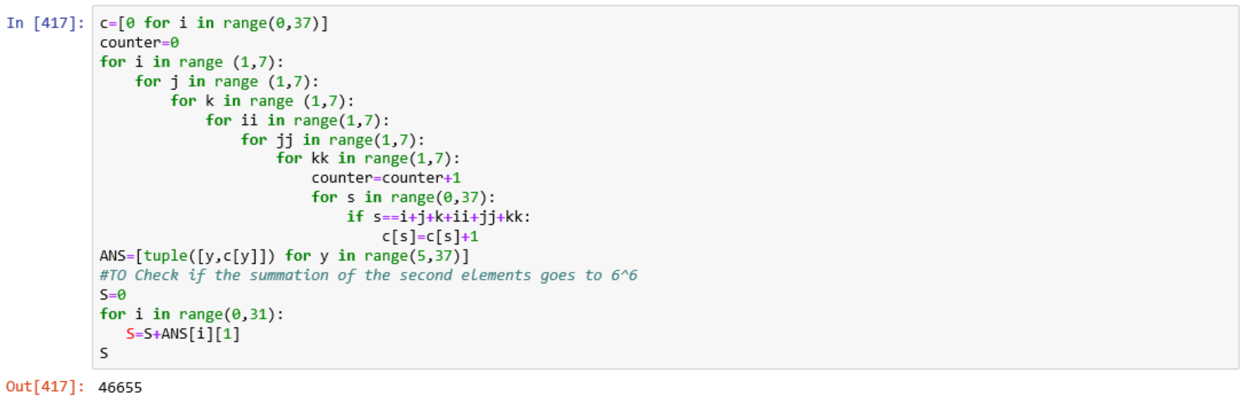
With the code of



```
In [415]: np.random.seed(20)
          asc = [tuple([np.random.randint(65,91),
          np.random.randint(65,91)]) for i in range(100)]
          a=[0 for i in range(100)]
          for i in range(100):
              a[i]=chr(asc[i][0])+chr(asc[i][1])
          a
```

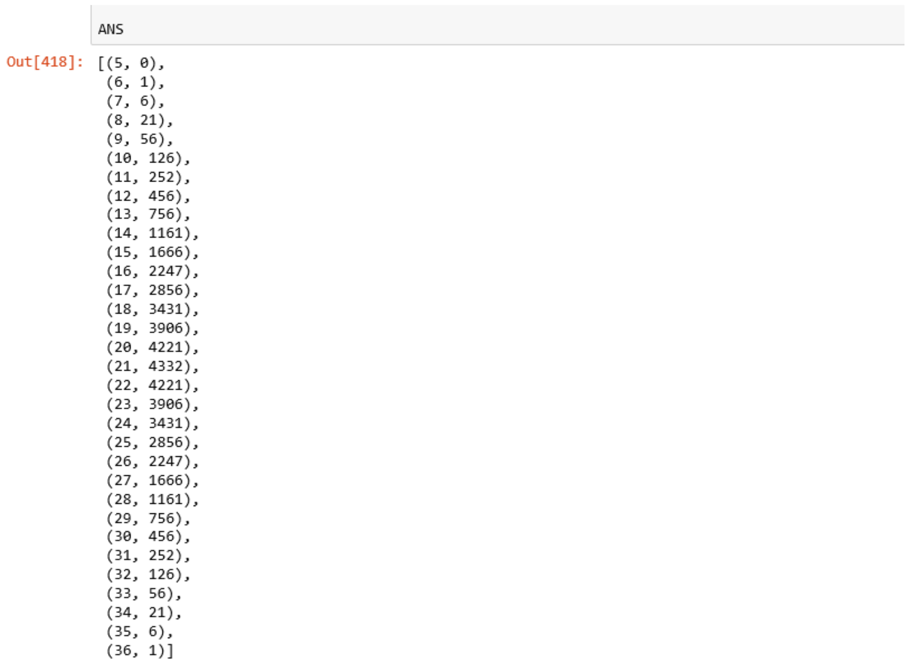
Q2.5)

In this question, the total number of combinations where 6 dices can, possibly, be thrown were calculated. A counter was used in the last of six nested for-loops. But for the total count of certain numbers, an array was defined where it has 36 elements. This array was used to store the resulted number (after throwing dices) to same address. The code is follows:



```
In [417]: c=[0 for i in range(0,37)]
          counter=0
          for i in range (1,7):
              for j in range (1,7):
                  for k in range (1,7):
                      for ii in range(1,7):
                          for jj in range(1,7):
                              for kk in range(1,7):
                                  counter=counter+1
                                  for s in range(0,37):
                                      if s==i+j+k+ii+jj+kk:
                                          c[s]=c[s]+1
          ANS=[tuple([y,c[y]]) for y in range(5,37)]
          #TO Check if the summation of the second elements goes to 6^6
          S=0
          for i in range(0,31):
             S=S+ANS[i][1]
          S

Out[417]: 46655
```

And ANS array is:

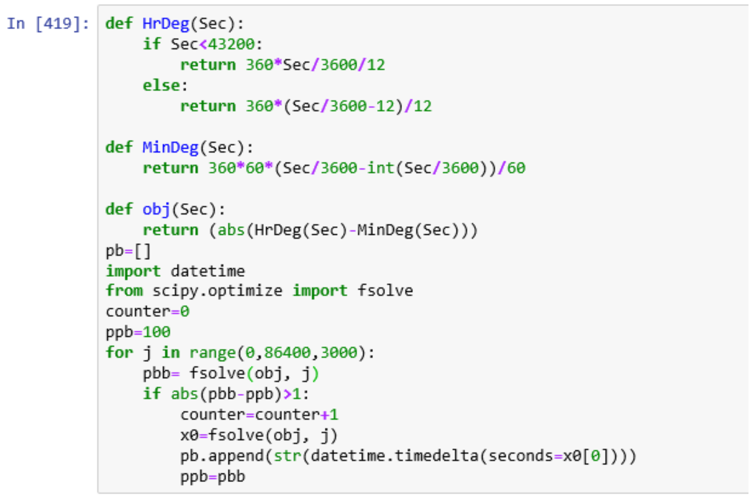


```
          ANS

Out[418]: [(5, 0),
           (6, 1),
           (7, 6),
           (8, 21),
           (9, 56),
           (10, 126),
           (11, 252),
           (12, 456),
           (13, 756),
           (14, 1161),
           (15, 1666),
           (16, 2247),
           (17, 2856),
           (18, 3431),
           (19, 3906),
           (20, 4221),
           (21, 4332),
           (22, 4221),
           (23, 3906),
           (24, 3431),
           (25, 2856),
           (26, 2247),
           (27, 1666),
           (28, 1161),
           (29, 756),
           (30, 456),
           (31, 252),
           (32, 126),
           (33, 56),
           (34, 21),
           (35, 6),
           (36, 1)]
```

These numbers were checked and they are correct.

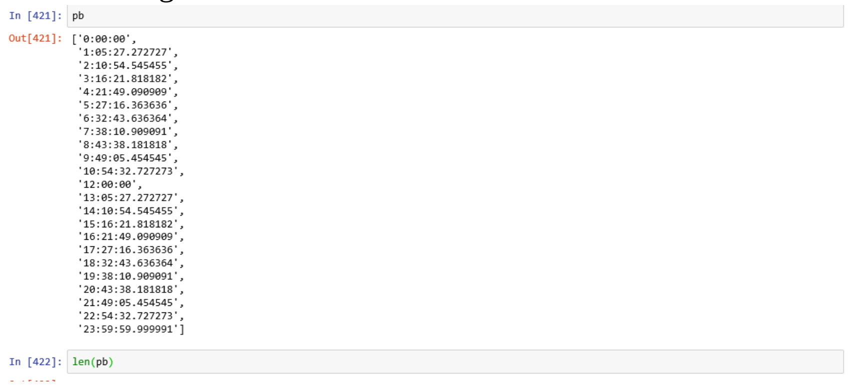
Q2.6)

In this question, two functions were defined where they give associated degrees for minute and hour hand based on Second. Then an objective function was defined where subtracts the two previous function values. Simply, if third function returns zero, it is believed that the corresponding second is the time of overlap.



```
In [419]: def HrDeg(Sec):
              if Sec<43200:
                  return 360*Sec/3600/12
              else:
                  return 360*(Sec/3600-12)/12

          def MinDeg(Sec):
              return 360*60*(Sec/3600-int(Sec/3600))/60

          def obj(Sec):
              return (abs(HrDeg(Sec)-MinDeg(Sec)))
          pb=[]
          import datetime
          from scipy.optimize import fsolve
          counter=0
          ppb=100
          for j in range(0,86400,3000):
              pbb= fsolve(obj, j)
              if abs(pbb-ppb)>1:
                  counter=counter+1
                  x0=fsolve(obj, j)
                  pb.append(str(datetime.timedelta(seconds=x0[0])))
                  ppb=pbb
```

The following is the answer:



```
In [421]: pb

Out[421]: ['0:00:00',
           '1:05:27.272727',
           '2:10:54.545455',
           '3:16:21.818182',
           '4:21:49.090909',
           '5:27:16.363636',
           '6:32:43.636364',
           '7:38:10.909091',
           '8:43:38.181818',
           '9:49:05.454545',
           '10:54:32.727273',
           '12:00:00',
           '13:05:27.272727',
           '14:10:54.545455',
           '15:16:21.818182',
           '16:21:49.090909',
           '17:27:16.363636',
           '18:32:43.636364',
           '19:38:10.909091',
           '20:43:38.181818',
           '21:49:05.454545',
           '22:54:32.727273',
           '23:59:59.999991']

In [422]: len(pb)
```

The answers were checked with the following link:

https://www.helpingwithmath.com/printables/worksheets/time/3md1-clock-face-generator01.htm