python_coding_test

August 14, 2018

```
In [1]: import numpy as np
        import time
        import pandas as pd
        import string
        import random

        np.random.seed(12345)
```

```
In [25]: %load_ext memory_profiler
```

1) QUESTION Sort the list below

```
In [35]: nump1 = np.random.randint(0, 10000, 1000000)
         list1 = list(nump1)
```

```
In [ ]: # Sort in-place
        list1.sort()

        # Create a new sorted list
        sorted(list1)

        # Numpy array (for memory usage and speed)
        np.sort(nump1)
```

2) QUESTION Given the list below of random length, fetch the last 10 elements of the list

```
In [20]: list1 = list(np.random.randint(0, 100, np.random.randint(50,100,1)))
```

```
In [21]: list1[-10:]
```

```
Out[21]: [57, 33, 18, 51, 83, 38, 13, 62, 99, 24]
```

3) QUESTION Given the list below find the indices of those elements greater than 80

```
In [61]: nump1 = np.random.randint(0, 100, 100)
         list1 = list(nump1)
```

```
In [62]: # List
         indices = [index for index,value in enumerate(list1) if value > 80]

         # With numpy array
         np.where(nump1 > 80)
```

```
Out[62]: (array([ 4, 23, 30, 36, 40, 43, 50, 61, 62, 65, 67, 80, 83, 87, 98, 99], c
```

4) QUESTION Given the list of strings below sort the list by string length

```
In [67]: list1 = ["".join(random.sample(string.ascii_lowercase*10, x)) for x in ran
         random.shuffle(list1)
```

```
In [68]: # I used key parameter to specify a function to be called on each list ele
         sorted(list1, key=len)
```

```
Out[68]: ['',
          'y',
          'ez',
          'cfl',
          'lumt',
          'ktuwg',
          'ojeptp',
          'igmtbae',
          'jimdbmwv',
          'wlaquqerz',
          'upbrbkgecq',
          'cevptkvxznk',
          'brqlpoeezjpt',
          'zciepyuxfaqwg',
          'rcpawttnlybghq']
```

5) QUESTION Given the two lists below find the intersection between both lists

```
In [77]: nump1 = np.random.randint(0, 100, 30)
         nump2 = np.random.randint(0, 100, 30)
         list1 = list(nump1)
         list2 = list(nump2)
```

```
In [79]: # here it's better to use a set to compute math operations like intersecti
         set(list1).intersection(list2)

         # With numpy arrays
         np.intersect1d(nump1, nump2)
```

```
Out[79]: {6, 40, 41, 85}
```

6) QUESTION Transform the two lists below into a list of tuples

```
In [83]: list1 = list(np.random.randint(0, 100, 10))
         list2 = list(np.random.randint(0, 100, 10))
```

```
In [84]: list(zip(list1, list2))

Out[84]: [(74, 54),
          (41, 66),
          (35, 83),
          (9, 78),
          (34, 88),
          (27, 25),
          (24, 87),
          (82, 52),
          (7, 61),
          (89, 90)]
```

7) QUESTION Convert the two lists below into a dictionary where `names` are keys and `numbers` are values

```
In [86]: names = ['One', 'Two', 'Three', 'Four', 'Five']
         numbers = [1, 2, 3, 4, 5]

In [90]: # If I use a list comprehension
         dic = {key:value for key, value in zip(names, numbers)}

         # Better
         dict(zip(names, numbers))

Out[90]: {'Five': 5, 'Four': 4, 'One': 1, 'Three': 3, 'Two': 2}
```

8) QUESTION Given the dictionary below, remove the element with key `One`

```
In [91]: dict1 = {'Four': 4, 'Five': 5, 'Three': 3, 'Two': 2, 'One': 1}

In [96]: dict2 = {key:value for key, value in dict1.items() if key != 'One'}
         dict2

Out[96]: {'Five': 5, 'Four': 4, 'Three': 3, 'Two': 2}
```

9) QUESTION Given the dictionary below, remove all elements with value `1` (do not use the key to do it)

```
In [98]: dict1 = {'Four': 4, 'Five': 5, 'Three': 3, 'Two': 2, 'One': 1, 'Uno':1, 'U

In [100]: dict2 = {key:value for key, value in dict1.items() if value != 1}
          dict2

Out[100]: {'Five': 5, 'Four': 4, 'Three': 3, 'Two': 2}
```

10) QUESTION How would you replace the following with a list comprehension ?

```
In [20]: first_ten_cubes = []
         for value in range(1, 11):
             cube = value*value*value
             first_ten_cubes.append(cube)
         first_ten_cubes
```

```
Out[20]: [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

```
In [103]: first_ten_cubes = [pow(i, 3) for i in range(1, 11)]
          first_ten_cubes

Out[103]: [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

11) QUESTION How would you sort the following dictionary by values ?

```
In [104]: d = dict(a=1, b=3, c=4, d=2)

In [110]: sorted(d.items(), key=lambda x:x[1])

Out[110]: [('a', 1), ('d', 2), ('b', 3), ('c', 4)]
```

12) QUESTION Implement a binary search algorithm returning the index of a given value in a sorted list in a function such as `binary_search([1, 2, 3, 4, 5], 3) = 2`

```
In [167]: def binary_search(liste, item):
              """Return indice of an item by applying a binary search"""

              low = 0
              high = len(liste) - 1

              guess = 0

              while low <= high:
                  index = int((low + high) / 2)
                  guess = liste[index]
                  if item > guess :
                      low = index + 1
                  elif item < guess :
                      high = index - 1
                  elif item == guess :
                      return index

In [170]: binary_search([1, 2, 3, 4, 5,6,26,89,102], 1)

Out[170]: 0
```

13) QUESTION Write a function that determines if a word is a palyndrome (it is read equally forthwards and backwards, eg: Hannah)

```
is_palyndrome("hannah") -> True
is_palyndrome("Hannah") -> True
is_palyndrome("Montana") - > False
```

```
In [122]: def is_palyndrome(word):
              """Check if a word is a palyndrome"""
              return word.lower() == word[::-1].lower()
```

```
In [124]: is_palyndrome("Hannah")

Out[124]: True
```

14) **QUESTION** Optimise the following code for performance and readability. Comment on the main problems found in the code and implement a more efficient version giving the same results in under 5 seconds. (You can use either Python built-in functions or any other library)

```
In [156]: matrix = np.random.rand(10000, 10000)
          n_rows, n_cols = matrix.shape

          start = time.time()
          row_averages = []
          for i in range(n_rows):
              row_averages = row_averages + [0]
              for j in range(n_cols):
                  row_averages[i] = row_averages[i] + matrix[i, j]
              row_averages[i] = row_averages[i] / n_cols
          print("The standard deviation across row averages is {}".format(np.std(ro
          col_averages = []
          for j in range(n_cols):
              col_averages = col_averages + [0]
              for i in range(n_rows):
                  col_averages[j] = col_averages[j] + matrix[i, j]
              col_averages[j] = col_averages[j] / n_rows
          print("The standard deviation across column averages is {}".format(np.std
          end = time.time()
          print("The analysis took {} secs".format(end-start))

The standard deviation across row averages is 0.002879392870208019
The standard deviation across column averages is 0.0028183027121602426
The analysis took 134.05800008773804 secs


In [154]: matrix = np.random.rand(10000, 10000)

          start = time.time()
          print("The standard deviation across row averages is {}".format(np.std(np
          print("The standard deviation across column averages is {}".format(np.std
          end = time.time()
          print("The analysis took {} secs".format(end-start))

The standard deviation across row averages is 0.0029249752740803723
The standard deviation across column averages is 0.0028739010093218015
The analysis took 0.2999999523162842 secs
Wall time: 2.1 s
```