algorithmica

Programming Language Learning Series
Mastery of Python Language
(Interview Questions/Assignment-List,Tuple)

Q1: Create a list with the values 1,2,4,6,6,9,10,11,14. Use indexing to

- print the 3rd value
- print the last 2 values
- print only the odd values

Q2: By using list comprehension, please write a program for the following:

- print the list after removing numbers which are divisible by 3 and 5 in [5, 10, 40, 7, 53, 31, 12, 101]
- print the list after removing the 0th, 2nd, 4th,6th numbers in [5, 10, 40, 7, 53, 31, 12, 101]
- generate a 3 * 4 2D-list whose elements are initialized to 0
- generate 2*3*5 3D-list whose each elements are initialized to 0

Q3: Write a function to generate all possible sentences, using list comprehension, that can be formed using following lists: ["Python", "Cpp", “Scala”], ["learning", "language"], ["beautiful","fun"]. Here, you will get 12 possible sentences in total.

Q4: Write a function which finds all numbers between 1000 and 3000(both included) such that each digit of the number is an even digit. The numbers obtained should be printed in a comma-separated sequence on a single line.

Q5: Write a function that accepts a comma separated sequence of words as input and prints the words in a comma-separated sequence after sorting them alphabetically.
Input: education, is, experience, joyful, fun, not, mess
Output: education, experience, fun, is, joyful, mess, not

Q6: Write a function that can flatten a nested list of arbitrary depth.
Input: [1,[2,3],[4,[5,[6,7],8],9],10,[11,12]])
Output: [1,2,3,4,5,6,7,8,9,10,11,12]

Q7: Write a function to sort the list of (name, age, height) tuples by ascending order where name is string, age and height are numbers. The sort criteria is:

1. Sort based on name
2. Then sort based on age
3. Then sort by score

The priority is that name > age > score

Input: [(‘Tagore’,19,80), (‘Joseph’,20,90), (‘Jony’,17,91), (‘Joseph’,17,94), (‘Aurobindo’,21,85)]
Output: [(‘Aurobindo’,21,85), (‘Jony’,17,91), (‘Joseph’,17,94), (‘Joseph’,20,90), (‘Tagore’,19,80)]

Q8: A robot moves in a plane starting from the original point (0,0). The robot can move toward UP, DOWN, LEFT and RIGHT with a given steps. The numbers after the direction are steps. Write a program to compute the distance between the position after a sequence of movements and original point. If the distance is a float, then just print the nearest integer.
Input: [(‘UP’,5), (‘DOWN’,3), (‘LEFT’,3), (‘RIGHT’,2)]
Output: 2

Q9: Which of the following function is faster? Why?

```
def concat1(alist):
   """Using string concatenation."""
   s = alist[0]
   for item in alist[1:]:
      s += " " + item
   return s

def concat2(alist):
   """Using join."""
   return " ".join(alist)
```

```
def h(n, xs=[]):
   for i in range(n):
      xs.append(i)
   return xs
n = 5
xs = [1,2,3]
h(n)
h(n)
```

shallow vs deep copy example
i) Reference the original object. This points a new name, li2, to the same place in memory to which li1 points. So any change we make to li1 also occurs to li2.

li1 = [['a'],['b'],['c']]
li2 = li1li1.append(['d'])
print(li2)
#=> [['a'], ['b'], ['c'], ['d']]

ii) Create a shallow copy of the original. We can do this with the list() constructor, or the more pythonic mylist.copy() (thanks Chrisjan Wust !).

A shallow copy creates a new object, but fills it with references to the original. So adding a new object to the original collection, li3, doesn't propagate to li4, but modifying one of the objects in li3 will propagate to li4.

li3 = [['a'],['b'],['c']]
li4 = list(li3)li3.append([4])
print(li4)
#=> [['a'], ['b'], ['c']]li3[0][0] = ['X']
print(li4)
#=> [[['X']], ['b'], ['c']]

iii) Create a deep copy. This is done with copy.deepcopy(). The 2 objects are now completely independent and changes to either have no affect on the other.

import copyli5 = [['a'],['b'],['c']]
li6 = copy.deepcopy(li5)li5.append([4])
li5[0][0] = ['X']
print(li6)
#=> [['a'], ['b'], ['c']]

list slicing is copy/view