Version 1

**General Notes**:

* This document is being extended constantly. Look for the version number at the top.
* Many questions have several possible valid solutions. Here we propose just one.
* The questions are divided into subjects for your convenience, though some of the questions combine several subjects.
* It is advised to try and answer the questions yourself before reading the solution.

**Python Basics, Lists, Strings and Functions**:

**Question 1:**

1. Shortly explain the difference between a compiler and an interpreter.
2. A Python program consists of 20 lines, all in the \_\_main\_\_ scope. There are print() calls on lines 8, 13 and 20. Line 18 consists of an error that will raise an exception. Will the print() commands be executed? If so, which of them?

**Solution 1:**

1. A compiler takes an entire program written in a high-level programming language and translates it into machine code. An interpreter translates a program written in a high-level programming language to machine code line-by-line. Compilers can find (some) errors in advance, while an interpreter will only find errors during runtime.
2. Since an interpreter runs the program line-by-line, the print commands at lines 8 and 13 will be executed, and the one on line 20 will not because the exception prevented the interpreter to reach it**.**

**Question 2:**

Define a function overlapping() that takes two lists and returns a list of all the members they have in common. Do the exercise twice:

* With two nested for loops.
* With a single loop using the ‘in’ operator inside the loop

**Solution 2:**

**def** overlapping(lst1,lst2):  
 common = []  
 **for** element1 **in** lst1:  
 **for** element2 **in** lst2:  
 **if** element1 == element2:  
 common.append(element1)  
 **return** common  
  
**def** overlappingSecondVersion(lst1,lst2):  
 common = []  
 **for** element **in** lst1:  
 **if** element **in** lst2:  
 common.append(element)  
 **return** common  
  
lst1 = [1,2,3,4]  
lst2 = [3,4,5,6,7]  
lst3 = overlapping(lst1,lst2)  
print(lst3)  
lst3 = overlappingSecondVersion(lst1,lst2)  
print(lst3)

**Question 3:**

Define a function IsPalindrome() that recognizes palindromes (i.e. words that look the same written backwards). For example, IsPalindrome("radar") should return True. Use only one index variable in your loop.

**Solution 3:**

**def** IsPalindrome(text):  
 **for** i **in** range(len(text)//2):  
 **if** text[i] != text[-(i+1)]:  
 **return False  
 return True**print(IsPalindrome(**"radar"**))  
print(IsPalindrome(**"raard"**))

**Question 4:**

Write a program which will find all such numbers which are divisible by 7 but are not a multiple of 5, between 2000 and 3200 (both included). The numbers obtained should be printed in a comma-separated sequence on a single line (hint: use the join function).

**Solution 4:**

l=[]  
**for** i **in** range(2000, 3201):  
 **if** (i%7 == 0) **and** (i%5 != 0):  
 l.append(str(i))  
  
outputString = **','**.join(l)  
print(outputString)

**Question 5:**

Write a recursive function that takes a number and prints the “99 bottles” song with the following format:

99 bottles of beer on the wall, 99 bottles of beer.  
Take one down, pass it around, 98 bottles of beer on the wall.

**Solution 5:**

**def** songGenerator(n):  
 **if** n == 0:  
 **return  
 if** n == 1:  
 print (n, **" bottle of beer on the wall,"**,n,**"bottle of beer"**)  
 print (**"Take it down, pass it around,"**,n-1,**"bottles of beer on the wall\n"**)  
 **else**:  
 print (n, **" bottles of beer on the wall,"**,n,**"bottles of beer"**)  
 print (**"Take one down, pass it around,"**,n-1,**"bottles of beer on the wall\n"**)  
 songGenerator(n-1)  
  
songGenerator(3)

**Question 6:**

Write a function that gets a list of strings and return the longest string. The list can also contain other lists of strings, so you need to look inside those lists as well. **Hint**: Use the *isinstance* function.

**Solution 6:**

**def** longest(l):  
 longestWord = **""  
 for** element **in** l:  
 **if** isinstance(element,list):  
 longest\_in\_inner\_list = longest(element)  
 **if** len(longest\_in\_inner\_list) > len(longestWord):  
 longestWord = longest\_in\_inner\_list  
 **else**:  
 **if** len(element) > len(longestWord):  
 longestWord = element  
 **return** longestWord  
  
lst = [**'one'**,**'two'**,[**'three'**,[**'four'**,**'five'**]]]

print(longest(lst))

**List Comprehensions:**

**Question 1:**

Write a single-line command that maps a list of words into a list of integers representing the lengths of the corresponding words.

**Solution 1:**

list = [**'tiny'**,**',medium'**,**'gigantic'**]  
lengths = [len(word) **for** word **in** list]  
print(lengths)

**Question 2:**

Now take another list of words and create a list by concatenating each element in the first list with each element in the second list.

**Solution 2:**

lst = [**'tiny'**,**',medium'**,**'gigantic'**]  
lst2 = [**'mouse'**,**'dog'**,**'elephant'**]  
concatenated = [x+**'\_'**+y **for** x **in** lst **for** y **in** lst2]  
print(concatenated)

**Question 3:**

Write a single-line function *twoLetterSubs* that gets a list of words and returns a list of all two-letter substrings that appear in those words. For example: twoLetterSubs(['hello','world']) should return ['he', 'el', 'll', 'lo', 'wo', 'or', 'rl', 'ld'].

**Solution 3:**

**def** twoLetterSubs(lst):  
 **return** [word[i:i+2] **for** word **in** lst **for** i **in** range(len(word)-1)]

**Question 4:**

Write a function Nahman() that gets a list lst, and by using list comprehensions returns the following list: [lst[0], lst[0:1], lst[0:2],....., lst[0:(n-1)], where n is the length of lst.

For instance, if lst = ['n', 'a', 'h', 'm', 'a', 'n', "meuman"] then the output should be:

['n', 'n', 'a', 'n', 'a', 'h', 'n', 'a', 'h', 'm', 'n', 'a', 'h', 'm', 'a', 'n', 'a', 'h', 'm', 'a', 'n']

**Solution 4:**

**def** Nahman(lst):  
 **return** [x **for** i **in** range(len(lst)) **for** x **in** lst[0:i+1]]

print(Nahman([**'n'**, **'a'**, **'h'**, **'m'**, **'a'**, **'n'**, **"meuman"**]))

**Tuples, Sets and Dictionaries:**

**Question 1:**

Create a program that reads triplets of inputs from the user until the value 0 is read as the third input. The program will create a list of tuples representing cubes and will print that list. Then it will create a list of the cube’s volumes and print that list as well. Try to write the shortest code when creating tuples and reading their values.

**Solution 1:**

x,y,z = (int(input(**"-->"**)),int(input(**"-->"**)),int(input(**"-->"**)))  
cubes = []  
**while** (z != 0):  
 tuple = (x,y,z)  
 cubes.append(tuple)  
 x,y,z = (int(input(**"-->"**)),int(input(**"-->"**)),int(input(**"-->"**)))  
print(cubes)  
  
volumes = []  
**for** cube **in** cubes:  
 x,y,z = cube  
 volumes.append(x\*y\*z)  
print(volumes)

**Question 2:**

Write a program which accepts a sequence of comma-separated numbers from the console and generates a list and a tuple which contains every number.

**Solution 2:**

values=input(**"-->"**)  
l=values.split(**","**)  
t=tuple(l)  
print (l)  
print (t)

**Question 3:**

Create a program that reads 5 **different** words from the user. Then the program will print a few empty lines and ask the user to provide the same words again from memory, not necessarily in the same order. The program will print out the words that the user remembered, the words that he forgot, and the excessive words.

**Solution 3:**

print(**"Please enter 5 different words"**)  
wordsSet = set()  
**while** len(wordsSet) < 5:  
 wordsSet.add(input(**"-->"**))  
print (**"\n\n\n\n\n\n\n"**)  
  
print(**"Now, please enter the same 5 words from memory, not necessarily in the same order"**)  
wordsSet2 = set()  
**while** len(wordsSet2) < 5:  
 wordsSet2.add(input(**"-->"**))  
  
print(**"You remembered the words:"**,wordsSet & wordsSet2)  
print(**"You forgot the words:"**,wordsSet - wordsSet2)  
print(**"The following words:"**,wordsSet2 - wordsSet,**"are excessive"**)

**Question 4:**

Write a function numberToDict() that gets a number and returns a dictionary mapping its digits location to their value, going from right to left. Then write the opposite function dictToNumber().

**Solution 4:**

**def** numberToDict(num):  
 d = {}  
 location = 0  
 **while**(num > 0):  
 d[location] = num % 10  
 num = num // 10  
 location += 1  
 **return** d  
  
**def** dictToNumber(d):  
 number = 0  
 **for** key,value **in** d.items():  
 number += 10 \*\* key \* value  
 **return** number  
  
d = numberToDict(85432)  
print(d)  
number = dictToNumber(d)  
print(number)

**Question 5:**

What are the similarities and the differences between a list and a tuple?

**Solution 5:**

Tuples and lists are both ordered sequences that can hold any type of elements. They can be accessed by an index and looped in a certain order. Tuples, unlike lists are immutable and we cannot change a tuple's element once it is created. Lists are mutable and have several methods and operators for adding, removing and changing the list's elements.