'''''

252. Meeting Rooms

Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],...] (si < ei), determine if a person could attend all meetings.

For example,

Given [[0, 30],[5, 10],[15, 20]],

return false.

'''

# Solution is to sort the array first using start time

# since intervals are not sorted and then compare end time of one meeting with end time

# of another meeting

2. # Definition for an interval.
3. # class Interval(object):
4. #     def \_\_init\_\_(self, s=0, e=0):
5. #         self.start = s
6. #         self.end = e
8. **class** Solution(object):

11. **def** sortInterval(self,intervals2):
12. **def** merge(a,b):
13. c = []
14. **while** len(a) > 0 **and** len(b) > 0:
15. **if** a[0].start > b[0].start:
16. c.append(b[0])
17. b.remove(b[0])
18. **else**:
19. c.append(a[0])
20. a.remove(a[0])
21. **if** len(a) == 0:
22. c = c + b
23. **else**:
24. c = c + a
25. **return** c
27. **if** len(intervals2) == 1 **or** len(intervals2) == 0:
28. **return** intervals2
29. #print len(intervals2)
30. mid = int(len(intervals2)/2)
31. #print "mid:", mid
32. a = self.sortInterval(intervals2[:mid])
33. b = self.sortInterval(intervals2[mid:])
34. **return** merge(a,b)

37. **def** canAttendMeetings(self, intervals):
38. """
39. :type intervals: List[Interval]
40. :rtype: bool
41. """
42. # sort the input using MergeSort Algo
43. intervals = self.sortInterval(intervals)
44. **for** i **in** range(1,len(intervals)):
45. **if** intervals[i-1].end > intervals[i].start:
46. **return** False
47. **return** True

'''''

293. Flip Game

You are playing the following Flip Game with your friend: Given a string that contains only these two characters: + and -, you and your friend take turns to flip two consecutive "++" into "--". The game ends when a person can no longer make a move and therefore the other person will be the winner.

Write a function to compute all possible states of the string after one valid move.

For example, given s = "++++", after one move, it may become one of the following states:

[

  "--++",

  "+--+",

  "++--"

]

If there is no valid move, return an empty list [].

'''

2. **class** Solution(object):
3. **def** generatePossibleNextMoves(self, s):
4. """
5. :type s: str
6. :rtype: List[str]
7. """
8. res = ""
9. arr = []
10. **for** i **in** range(len(s)-1):
11. res = s
12. **if** s[i] == '+' **and** s[i+1] == '+':
13. res = res[0:i] + '--' + res[i+2:]
14. arr.append(res)
15. **return** arr

'''''

412. Fizz Buzz

Write a program that outputs the string representation of numbers from 1 to n.

But for multiples of three it should output ‚ÄúFizz‚Äù instead of the number and for the multiples of five output ‚ÄúBuzz‚Äù. For numbers which are multiples of both three and five output ‚ÄúFizzBuzz‚Äù.

Example:

n = 15,

Return:

[

    "1",

    "2",

    "Fizz",

    "4",

    "Buzz",

    "Fizz",

    "7",

    "8",

    "Fizz",

    "Buzz",

    "11",

    "Fizz",

    "13",

    "14",

    "FizzBuzz"

]

'''

2. **class** Solution(object):
3. **def** fizzBuzz(self, n):
4. """
5. :type n: int
6. :rtype: List[str]
7. """
8. arr = []
9. i = 1
10. **while** i <= n:
11. **if** i%3 == 0 **and** i%5 == 0:
12. arr.append("FizzBuzz")
13. **elif** i%3 == 0:
14. arr.append("Fizz")
15. **elif** i%5 == 0:
16. arr.append("Buzz")
17. **else**:
18. arr.append(str(i))
19. i = i + 1
20. **return** arr

'''''

359. Logger Rate Limiter

Design a logger system that receive stream of messages along with its timestamps, each message should be printed if and only if it is not printed in the last 10 seconds.

Given a message and a timestamp (in seconds granularity), return true if the message should be printed in the given timestamp, otherwise returns false.

It is possible that several messages arrive roughly at the same time.

Example:

Logger logger = new Logger();

// logging string "foo" at timestamp 1

logger.shouldPrintMessage(1, "foo"); returns true;

// logging string "bar" at timestamp 2

logger.shouldPrintMessage(2,"bar"); returns true;

// logging string "foo" at timestamp 3

logger.shouldPrintMessage(3,"foo"); returns false;

// logging string "bar" at timestamp 8

logger.shouldPrintMessage(8,"bar"); returns false;

// logging string "foo" at timestamp 10

logger.shouldPrintMessage(10,"foo"); returns false;

// logging string "foo" at timestamp 11

logger.shouldPrintMessage(11,"foo"); returns true;

'''

2. **class** Logger(object):
4. messageTable = {}
5. **def** \_\_init\_\_(self):
6. """
7. Initialize your data structure here.
8. """
9. self.messageTable = {}


13. **def** shouldPrintMessage(self, timestamp, message):
14. """
15. Returns true if the message should be printed in the given timestamp, otherwise returns false.
16. If this method returns false, the message will not be printed.
17. The timestamp is in seconds granularity.
18. :type timestamp: int
19. :type message: str
20. :rtype: bool
21. """
23. # we will maintain a Hash table which will store the values for messages
24. **if** message **in** self.messageTable:
25. # if there is a message which is less than 10 seconds older dont print it
26. **if** self.messageTable[message] > timestamp - 10:
27. **return** False
29. # if there is no such message in last 10 second than store the value and print it
30. self.messageTable[message] = timestamp
31. **return** True
33. '''''

136. Single Number

Given an array of integers, every element appears twice except for one. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

'''

2. # my solution - use Hash table to store the counts of every element in the array
3. # and then check the hash table where it is equal to 1
4. **class** Solution(object):
5. **def** singleNumber(self, nums):
6. """
7. :type nums: List[int]
8. :rtype: int
9. """
10. hashNum = {}
11. **for** i **in** range(len(nums)):
12. **if** nums[i] **in** hashNum:
13. hashNum[nums[i]] += 1
14. **else**:
15. hashNum[nums[i]] = 1
16. **for** i **in** range(len(nums)):
17. **if** hashNum[nums[i]] == 1:
18. **return** nums[i]

'''''

Approach #2 Hash Table [Accepted]

Algorithm

We use hash table to avoid the O(n)O(n) time required for searching the elements.

Iterate through all elements in \text{nums}nums

Try if hash\\_tablehash\_table has the key for pop

If not, set up key/value pair

In the end, there is only one element in hash\\_tablehash\_table, so use popitem to get it

Python

'''

1. **class** Solution(object):
2. **def** singleNumber(self, nums):
3. """
4. :type nums: List[int]
5. :rtype: int
6. """
7. hash\_table = {}
8. **for** i **in** nums:
9. **try**:
10. hash\_table.pop(i)
11. **except**:
12. hash\_table[i] = 1
13. **return** hash\_table.popitem()[0]

'''''

Complexity Analysis

Time complexity : O(n \* 1) = O(n)O(n‚àó1)=O(n). Time complexity of for loop is O(n)O(n). Time complexity of hash table(dictionary in python) operation pop is O(1)O(1).

Space complexity : O(n)O(n). The space required by hash\\_tablehash\_table is equal to the number of elements in \text{nums}nums.

'''

'''''

Approach #3 Math [Accepted]

Concept

2 \* (a + b + c) - (a + a + b + b + c) = c2‚àó(a+b+c)‚àí(a+a+b+b+c)=c

Python

'''

2. **class** Solution(object):
3. **def** singleNumber(self, nums):
4. """
5. :type nums: List[int]
6. :rtype: int
7. """
8. **return** 2 \* sum(set(nums)) - sum(nums)

'''''

Complexity Analysis

Time complexity : O(n + n) = O(n)O(n+n)=O(n). sum will call next to iterate through \text{nums}nums. We can see it as sum(list(i, for i in nums)) which means the time complexity is O(n)O(n) because of the number of elements(nn) in \text{nums}nums.

Space complexity : O(n + n) = O(n)O(n+n)=O(n). set needs space for the elements in nums

'''



'''''

Approach #4 Bit Manipulation [Accepted]

Concept

If we take XOR of zero and some bit, it will return that bit

a \oplus 0 = aa‚äï0=a

If we take XOR of two same bits, it will return 0

a \oplus a = 0a‚äïa=0

a \oplus b \oplus a = (a \oplus a) \oplus b = 0 \oplus b = ba‚äïb‚äïa=(a‚äïa)‚äïb=0‚äïb=b

So we can XOR all bits together to find the unique number.

Python

'''

2. **class** Solution(object):
3. **def** singleNumber(self, nums):
4. """
5. :type nums: List[int]
6. :rtype: int
7. """
8. a = 0
9. **for** i **in** nums:
10. a ^= i
11. **return** a

'''''

Complexity Analysis

Time complexity : O(n)O(n). We only iterate through \text{nums}nums, so the time complexity is the number of elements in \text{nums}nums.

Space complexity : O(1)O(1).

'''

'''''

This function calculates Fibonocci number at nth place using Dynamic Programming

'''

2. fibHash = {}
3. **def** fibDP(n):
4. **if** n == 1:
5. **return** 1
6. **if** n == 0:
7. **return** 0
8. **if** n **in** fibHash:
9. **print** "resued"
10. **return** fibHash[n]
12. # store the computed value in Hash table
13. # Memoization
14. fibHash[n-1] = fibDP(n-1)
15. fibHash[n-2] = fibDP(n-2)
17. **print** fibHash
18. **return** fibDP(n-1) + fibDP(n-2)
20. **print** fibDP(10)
22. # this function creates an array out of an integer
23. # this can also be used to reverse a number
25. **def** createArrayFromInt(n):
26. arr = []
27. **while** n > 10:
28. arr.append(int(n%10))
29. n = n/10
30. arr.append(int(n))
31. **return** arr
33. **print** createArrayFromInt(123)

36. ## Recursive version - own
37. **def** createArrayFromInt(n):
38. **if** n <= 10:
39. **return** str(int(n))
41. **return** str(int(n%10)) + createArrayFromInt(n/10)
42. '''''
43. Find maximum sum a subarray can have in an array
44. Kadane's Algorithm
45. '''
47. **def** maxSubArray(arr):
48. max\_so\_far = 0
49. max\_ending\_here = 0
51. **for** i **in** range(len(arr)):
52. max\_ending\_here = max\_ending\_here + arr[i]
54. **if** max\_ending\_here < 0:
55. max\_ending\_here = 0
56. **if** max\_so\_far < max\_ending\_here:
57. max\_so\_far = max\_ending\_here
59. **return** max\_so\_far

62. myArr = [-1,-2,10,-9,8,3,1,-8]
63. **print** maxSubArray(myArr)
64. # Iterative solution
65. **def** findGCD(a,b):
67. **while** a != b:
68. **if** a > b:
69. a = a - b
70. **else**:
71. b = b - a
72. **return** a
74. **print** findGCD(108,12)
76. # recursive solution
77. **def** findGCDRecur(a,b):
78. **if** a == 0 **or** b == 0:
79. **return** 0
80. **if** a == b:
81. **return** a
82. **if** a > b:
83. **return** findGCDRecur(a-b,b)
84. **else**:
85. **return** findGCDRecur(a,b-a)

'''''

One Away: There are three types of edits that can be performed on strings: insert a character, remove a character, or replace a character. Given two strings, write a function to check if they are one edit (or zero edits) away.

EXAMPLE

pale, ple -> true

pales, pale -> true

pale, bale -> true

pale, bake -> false

'''

2. **def** oneEditAway(str1,str2):
3. hash1 = {}
4. hash2 = {}
5. differ = 0
6. # create hash table of both the strings and then compare them
7. **for** i **in** range(len(str1)):
8. **if** str1[i] **in** hash1:
9. hash1[str1[i]] += 1
10. **else**:
11. hash1[str1[i]] = 1
12. **for** i **in** range(len(str2)):
13. **if** str2[i] **in** hash2:
14. hash2[str2[i]] += 1
15. **else**:
16. hash2[str2[i]] = 1
18. **if** abs(len(hash1) - len(hash2)) > 1:
19. **return** False
20. **if** len(hash1) > len(hash2):
22. **for** word, times **in** hash1.items():
23. **if** word **not** **in** hash2:
24. differ +=1
25. **if** differ > 1 :
26. **return** False
27. **if** times > 1:
28. **return** False
29. **if** word **in** hash2:
30. **if** abs(hash1[word] - hash2[word]) > 1:
31. **return** False
32. **else**:
33. **for** word, times **in** hash2.items():
34. **if** word **not** **in** hash1:
35. differ +=1
36. **if** differ > 1 :
37. **return** False
38. **if** times > 1:
39. **return** False
40. **if** word **in** hash1:
41. **if** abs(hash2[word] - hash1[word]) > 1:
42. **return** False
43. **return** True
45. **print** oneEditAway('pales','pale')

'''''

88. Merge Sorted Array

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note:

You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2. The number of elements initialized in nums1 and nums2 are m and n respectively.

'''

2. **class** Solution(object):
3. **def** merge(self, nums1, m, nums2, n):
4. """
5. :type nums1: List[int]
6. :type m: int
7. :type nums2: List[int]
8. :type n: int
9. :rtype: void Do not return anything, modify nums1 in-place instead.
10. """
11. **while** n > 0 **and** m > 0:
12. **if** nums1[m-1] <= nums2[n-1]:
13. nums1[m+n-1] = nums2[n-1]
14. n = n -1
15. **else**:
16. nums1[m+n-1] = nums1[m-1]
17. m = m - 1
18. **print** "n:",n
19. **print** "m:",m
20. **print** "nums1:",nums1
21. **print** "nums2:",nums2
23. **if** n > 0:
24. #nums1 = nums2[0:n] + nums1[m+1:]
25. nums1[:n] = nums2[:n]
26. **print** nums1

'''''

521. Longest Uncommon Subsequence I

Given a group of two strings, you need to find the longest uncommon subsequence of this group of two strings. The longest uncommon subsequence is defined as the longest subsequence of one of these strings and this subsequence should not be any subsequence of the other strings.

A subsequence is a sequence that can be derived from one sequence by deleting some characters without changing the order of the remaining elements. Trivially, any string is a subsequence of itself and an empty string is a subsequence of any string.

The input will be two strings, and the output needs to be the length of the longest uncommon subsequence. If the longest uncommon subsequence doesn't exist, return -1.

Example 1:

Input: "aba", "cdc"

Output: 3

Explanation: The longest uncommon subsequence is "aba" (or "cdc"),

because "aba" is a subsequence of "aba",

but not a subsequence of any other strings in the group of two strings.

Note:

Both strings' lengths will not exceed 100.

Only letters from a ~ z will appear in input strings.

'''

2. **class** Solution(object):
3. **def** findLUSlength(self, a, b):
4. """
5. :type a: str
6. :type b: str
7. :rtype: int
8. """
9. **if** a == b:
10. **return** -1
11. **if** len(a) == len(b) **and** a != b:
12. **return** len(a)
13. **if** len(a) > len(b):
14. **return** len(a)
15. **return** len(b)

'''''

266. Palindrome Permutation

Given a string, determine if a permutation of the string could form a palindrome.

For example,

"code" -> False, "aab" -> True, "carerac" -> True.

'''

2. **class** Solution(object):
3. **def** canPermutePalindrome(self, s):
4. """
5. :type s: str
6. :rtype: bool
7. """
8. # a string is Palindrome of the each charater appears for even number of times
9. # and only character can appear for odd number of times
10. # so we count the characters using Hash map and then count
11. myLetters = {}
12. **for** i **in** range(len(s)):
13. **if** s[i] **in** myLetters:
14. #print s[i]
15. myLetters[s[i]] += 1
16. **else**:
17. myLetters[s[i]] = 1
18. #print myLetters
19. countOdd = 0
20. **for** k **in** myLetters:
21. #print myLetters[k]
22. **if** myLetters[k]%2 <> 0:
23. countOdd = countOdd + 1
24. #print "countOdd", countOdd
25. **if** countOdd > 1:
26. **return** False
27. **return** True
28. # this function converts decimal to binary for numbers upto 536,870,912
29. **def** convertToBinary(num):
30. binary\_table = []
31. b = 0
32. # create a table of powers of 2s
33. **while** b < 30:
34. binary\_table.append(2\*\*b)
35. b = b + 1
37. # this will be something like binary\_table = [1,2,4,8,16,32,64,128,256,512,1024...]
38. #instead I found a better way
39. binary\_table = [2\*\*b **for** b **in** range(30)]
40. my\_keys = {}
41. n = num
42. # try to find the value in binary\_table
43. # if found add it in the hash table
44. **while** n > 0:
45. i = 0
46. **while** n >= binary\_table[i]:
47. i = i + 1
49. **if** binary\_table[i-1] **in** my\_keys:
50. my\_keys[binary\_table[i-1]] += 1
51. **else**:
52. my\_keys[binary\_table[i-1]] = 1
53. n = n - binary\_table[i-1]
55. # my\_keys will be something like {4: 1, 8: 1, 16: 1, 32: 1, 64: 1}
56. i = 0
57. arr = []
58. # then create the binary numbers using keys
59. **while** 2\*\*i <= num:
60. **if** 2\*\*i **in** my\_keys:
61. arr.append(1)
62. **else**:
63. arr.append(0)
64. i = i + 1
65. **return** arr[::-1]
67. **print** convertToBinary(124)


71. **def** convertToDecimal(n):
72. arr = list(str(n))
73. **print** arr
74. i = len(arr) - 1
75. #print i
76. dec = 0
77. **while** i >= 0:
78. dec = dec + 2\*\*i\*int(arr[(len(arr)-i-1)])
79. #print i, 2\*\*i,dec
80. i = i - 1
81. **return** dec
83. **print** convertToDecimal('0101011010')

# Sorting an array or list

# Bubble sort

# Idea is to bubble up the largest (or smallest) item

# second largest, third largest and so on

# complexity O(n\*n)

1. **def** sortList(myList):
3. **for** i **in** range(len(myList)):
4. **for** j **in** range(len(myList) - 1 - i):
5. **if** myList[j] > myList[j + 1]:
6. # swap the tuple
7. (myList[j],myList[j + 1]) = (myList[j + 1], myList[j])
8. **return** myList
10. arr = [2,100,1,10,9,0,-1]
11. **print** sortList(arr)
13. # if it is a string
14. # do a minor change in above function
15. **def** sortList(myList):
17. **for** i **in** range(len(myList)):
18. **for** j **in** range(len(myList) - 1 - i):
19. **if** myList[j] > myList[j + 1]:
20. # swap the tuple
21. (myList[j],myList[j + 1]) = (myList[j + 1], myList[j])
22. **return** "".join(myList)
24. str = list("dbcaA")
25. **print** sortList(str)

28. # Insertion sort
29. # Complexity O(nlogn) , worst O(n\*n)
30. myArray = [3,1,4,5,10,2,7,100]
31. res = [myArray[0]]


35. **for** i **in** range(1,len(myArray)):
36. res.append(myArray[i])
37. j = i
38. **while** j > 0:
39. **if** res[j] < res[j-1]:
40. # "swapping"
41. (res[j],res[j-1]) = (res[j-1], res[j])
42. j = j -1
43. **else**:
44. **break**
46. **print** res

49. # Mergesort
50. # Divide and Conquer
51. # Complexity always O(nlogn)
52. **def** merge(a,b):
53. c = []
54. **while** len(a) > 0 **and** len(b) > 0:
55. **if** a[0] < b[0]:
56. c.append(a[0])
57. a.remove(a[0])
58. **else**:
59. c.append(b[0])
60. b.remove(b[0])
61. **if** len(a) == 0:
62. c = c + b
63. **else**:
64. c = c + b
65. **return** c
67. **def** mergeSort(myList):
69. **if** len(myList) == 1:
70. **return** myList
71. **else**:
73. middle = int(len(myList)/2)
75. a = mergeSort(myList[:middle])
76. b = mergeSort(myList[middle:])
77. **return** merge(a,b)


81. myArray = [3,1,4,2,7,9,10]
82. **print** mergeSort(myArray)




88. # script to check if string has unique characters
90. **def** isUniqueChar(str):
91. # if string has more than 128 characters than it definitely has duplicate characters
92. # since we have 128 unique characters in ascii
94. **if** len(str) > 128:
95. **return** False
97. # create an array for holding each of 128 unicode characters
98. # assign False to each initially
99. # then assign True while looping through the string if a charcter is found
100. # ord return unicode value of a character
101. arr = [False] \* 128
103. **for** each\_char **in** str:
104. **if** arr[ord(each\_char.lower())] **is** False:
105. arr[ord(each\_char.lower())] = True
106. **else**:
107. **return** False
109. **return** True

112. testString = "abcdA"
114. **print** "String " + testString + " has Unique characters? " + str(isUniqueChar(testString))

117. #similar approach using Hash table
118. **def** isUnique(str):
119. char\_Hash = {}
120. # create a hash table for all characters and assign them to False
121. **for** i **in** range(ord('a'),ord('z')+1):
122. char\_Hash[chr(i)] = False
124. # if value for char is found True then it is appearing twice
125. **for** i **in** range(len(str)):
126. **if** char\_Hash[str[i]] == True:
127. **return** False
128. **else**:
129. char\_Hash[str[i]] = True
131. **return** True
133. **print** isUnique('abdc'.lower())

136. # this prints last 2 characters of string
137. **def** extra\_end(str):
138. str2 = ""
139. **for** i **in** range(3):
140. str2 = str2 + str[-2:]
141. **return** str2

144. # first 2 characters
145. **def** first\_two(str):
146. **if** len(str) >=2:
147. **return** str[0:2]
148. **return** str

'''''

Return True if the string "cat" and "dog" appear the same number of times in the given string.

cat\_dog('catdog') ‚Üí True

cat\_dog('catcat') ‚Üí False

cat\_dog('1cat1cadodog') ‚Üí True

'''

1. **def** cat\_dog(str):
2. cat = 0
3. dog = 0
4. **for** i **in** range(2,len(str)):
5. **if** str[i] == 't' **and** str[i-1] == 'a' **and** str[i-2] == 'c':
6. cat = cat + 1
7. **if** str[i] == 'g' **and** str[i-1] == 'o' **and** str[i-2] == 'd':
8. dog = dog + 1
9. **if** cat == dog:
10. **return** True
11. **return** False



16. # return how many times string code appears in string
17. # except we'll accept any letter for the 'd', so "cope" and "cooe" count.
18. **def** count\_code(str):
19. code = 0
20. **for** i **in** range(3,len(str)):
21. **if** str[i] == 'e' **and** str[i-2] == 'o' **and** str[i-3] == 'c':
22. code = code + 1
23. **return** code

# compare end part of 2 strings

'''''

Given two strings, return True if either of the strings appears at the very end of the other string,

ignoring upper/lower case differences (in other words, the computation should not be "case sensitive").

Note: s.lower() returns the lowercase version of a string.

end\_other('Hiabc', 'abc') ‚Üí True

end\_other('AbC', 'HiaBc') ‚Üí True

end\_other('abc', 'abXabc') ‚Üí True

'''

1. **def** end\_other(a, b):
2. **if** len(a) > len(b) **and** a[-len(b)::].lower() == b.lower():
3. **return** True
4. **elif** len(b) > len(a) **and** b[-len(a)::].lower() == a.lower():
5. **return** True
6. **elif** a.lower() == b.lower():
7. **return** True
8. **return** False


12. '''''
13. Find first occurence of a character in a string
14. return -1 if not found
15. '''
16. **def** findNeedle(str,needle):
17. **for** i **in** range(len(str)):
18. **if** str[i] == needle:
19. **return** i
21. **return** -1
23. **print** findNeedle("mystring","k")


27. #simple loop for lists or arrays
28. **def** count\_evens(nums):
29. even = 0
30. **for** i **in** range(len(nums)):
31. **if** nums[i]%2 == 0:
32. even += 1
33. **return** even
35. #return the difference between smallest and largest number in array
36. **def** big\_diff(nums):
37. big = 0
38. small = 0
39. **for** i **in** range(len(nums)):
40. **if** i == 0:
41. big = nums[i]
42. small = nums[i]
43. **if** nums[i] > big:
44. big = nums[i]
45. **if** nums[i] < small:
46. small = nums[i]
48. **return** big - small
50. #return the sum of the array, dont count 13 and next element to 13
51. **def** sum13(nums):
52. sum = 0
53. **for** i **in** range(len(nums)):
54. **if** i == 0 **and** nums[i] != 13:
55. sum = sum + nums[i]
56. **elif** i > 0 **and** nums[i] != 13 **and** nums[i-1] != 13:
57. sum = sum + nums[i]
58. **return** sum


62. ## make choclate problem
63. **def** make\_chocolate(small, big, goal):
64. **if** small + (big\*5) < goal:
65. **return** -1
66. **elif** goal%5 > small:
67. **return** -1
68. **elif** goal < 5 **and** goal <= small:
69. **return** goal
70. **elif** goal%5 == 0 **and** big\*5 >= goal:
71. **return** 0
72. **elif** goal%5 == 0 **and** (big\*5 + small) >= goal:
73. **return** small - big
74. **elif** goal%5 <= small:
75. **return** goal - big\*5

##################  IF - ELSE logical operators ####################

2. **def** cigar\_party(cigars, is\_weekend):
3. **if** is\_weekend:
4. **if** cigars >= 40:
5. **return** True
6. **return** False
7. **if** cigars >= 40 **and** cigars <= 60:
8. **return** True
10. **return** False

13. '''''

Given 3 int values, a b c, return their sum. However, if one of the values is the same as another of the values, it does not count towards the sum.

2. lone\_sum(1, 2, 3) ‚Üí 6
3. lone\_sum(3, 2, 3) ‚Üí 2
4. lone\_sum(3, 3, 3) ‚Üí 0
5. '''
6. **def** lone\_sum(a, b, c):
7. **if** a != b **and** b != c **and** c != a:
8. **return** a+b+c
9. **elif** a == b **and** a != c:
10. **return** c
11. **elif** b == c **and** a != b:
12. **return** a
13. **elif** a == c **and** a!= b:
14. **return** b
15. **elif** a == b **and** b == c:
16. **return** 0
18. '''''

Given 3 int values, a b c, return their sum. However, if any of the values is a teen -- in the range 13..19 inclusive -- then that value counts as 0,

except 15 and 16 do not count as a teens.

Write a separate helper "def fix\_teen(n):"that takes in an int value and returns that value fixed for the teen rule. I

n this way, you avoid repeating the teen code 3 times (i.e. "decomposition"). Define the helper below and at the same indent level as the main no\_teen\_sum().

1. '''
2. **def** no\_teen\_sum(a, b, c):
3. **return** fix\_teen(a) + fix\_teen(b) + fix\_teen(c)
5. **def** fix\_teen(n):
6. **if** n >= 13 **and** n <= 19 **and** n != 15 **and** n != 16:
7. **return** 0
8. **else**:
9. **return** n

12. '''''

For this problem, we'll round an int value up to the next multiple of 10 if its rightmost digit is 5 or more, so 15 rounds up to 20.

Alternately, round down to the previous multiple of 10 if its rightmost digit is less than 5, so 12 rounds down to 10. Given 3 ints, a b c,

return the sum of their rounded values.

To avoid code repetition, write a separate helper "def round10(num):" and call it 3 times.

Write the helper entirely below and at the same indent level as round\_sum()

1. '''
2. **def** round\_sum(a, b, c):
3. **return** round10(a) + round10(b) + round10(c)
5. **def** round10(num):
6. **if** num%10 == 0:
7. **return** num
8. **elif** num%10 == 5:
9. **return** num+5
10. **elif** num%10 > 5:
11. **return** num - (num%10) + 10
12. **elif** num%10 < 5:
13. **return** num - (num%10)



18. ##makebricks problem
20. **def** make\_bricks(small, big, goal):
21. **if** small + (big\*5) == goal:
22. **return** True
23. **elif** small + (big\*5) < goal:
24. **return** False
25. **elif** goal%5 > small:
26. **return** False
27. ##if we remove big bricks from total and reach the goal
28. **elif** small + (big\*5) > goal **and** (((small + (big\*5)) - goal)%5 == 0):
29. **return** True
30. ##if we remove small bricks from total and reach the goal
31. **elif** small + (big\*5) > goal **and** ((small + (big\*5)) - goal) <= small:
32. **return** True
33. **return** True
34. #list operations
36. # return true if first and last element of the array is 6
37. **def** first\_last6(nums):
38. **if** len(nums) == 1 **and** nums[0] == 6:
39. **return** True
40. **elif** len(nums) > 1 **and** (nums[0] == 6 **or** nums[len(nums)-1] == 6):
41. **return** True
42. **return** False

45. # return True if first and last element of the array are equal and array size >= 1
46. **def** same\_first\_last(nums):
48. **if** len(nums) >= 1 **and** (nums[0] == nums[len(nums) - 1]):
49. **return** True
50. **return** False
52. #return True if they have the same first element or they have the same last element
53. **def** common\_end(a, b):
54. **if** a[0] == b[0] **or** a[len(a)-1] == b[len(b)-1]:
55. **return** True
56. **return** False

59. #return sum of all elements of array
60. **def** sum3(nums):
61. a = 0
62. **for** i **in** range(len(nums)):
63. a = a + nums[i]
64. **return** a
66. #concatenate 2 list of integers
67. **def** rotate\_left3(nums):
68. **return** nums[1:] + map(int, str(nums[0]))


72. #loop a string in reverse order
73. **def** reverse3(nums):
74. result = []
75. **for** i **in** range(len(nums)-1, -1, -1):
76. result.append(nums[i])
78. **return** result

81. #loop a list of ints
82. **def** max\_end3(nums):
83. result = []
84. **if** nums[0] > nums[len(nums)-1]:
85. **for** i **in** range(len(nums)):
86. result.append(nums[0])
87. **return** result
88. **for** i **in** range(len(nums)):
89. result.append(nums[len(nums)-1])
90. **return** result

93. #loop through a list and print sum of first 2 elements
94. **def** sum2(nums):
95. result = 0
96. **if** len(nums) == 0:
97. **return** 0
98. **for** i **in** range(len(nums)):
99. result = result + nums[i]
100. **if** i > 0:
101. **return** result
102. **return** result

105. #concatenate middle elements of 2 arrays
106. **def** middle\_way(a, b):
108. **return** [a[int(math.ceil(len(a)/2))], b[int(math.ceil(len(a)/2))]]

111. #return an array containing first and last elements of array
112. **def** make\_ends(nums):
113. **return** [nums[0], nums[len(nums)-1]]
115. #return True if an array contains 2 or 3
116. **def** has23(nums):
117. **for** i **in** range(len(nums)):
118. **if** nums[i] == 2 **or** nums[i] == 3:
119. **return** True
120. **return** False




126. # move zeroes to the end of the array without changing the relative position of other elements
127. arr = [0,1,0,3,12]
128. **for** i **in** range(len(arr)):
129. **if** arr[i] == 0:
130. arr.append(0)
131. arr.pop(i)
133. **print** arr
135. #more optimal approach
136. #find non zero elements, if found move them
137. #in the end fill the tail of the array with Zeroes
139. arr = [0,0,0,0,12]
140. lastNonZero = 0
142. **for** i **in** range(len(arr)):
143. **if** arr[i] != 0:
144. arr[lastNonZero] = arr[i]
145. lastNonZero += 1
146. '''''elif arr[i] != 0 and i > 0:
147. lastNonZero += 1
148. arr[lastNonZero] = arr[i]
149. '''
150. **for** i **in** range(lastNonZero,len(arr)):
151. arr[i] = 0
153. **print** arr
155. # this function checks if a string is permutation of other string

158. **def** checkPermutation(str1, str2):
159. **if** len(str1) != len(str2):
160. **return** False
161. **else**:
162. str1Sorted = sorted(str1.lower())
163. str2Sorted = sorted(str2.lower())
165. s=""
167. **if** s.join(str1Sorted) == s.join(str2Sorted):
168. **return** True
169. **else**:
170. **return** False
172. **print** str(checkPermutation("abc","DEF"))