hackerrank_

August 28, 2023

[4]:
```
from datetime import datetime
import math
import os
import random
import re
import sys
from datetime import datetime
from itertools import groupby, combinations
from collections import deque
import numpy as np
```

[133]:
```
for num in range(2, 2):
    print("Hello")
```

[27]:
```
##############################################3
def time_delta(t1, t2):
    # .strptime(date_string, format) => return a datetime corresponding to␣
 ↪date_string, parsed according to format.
    # %a (weekday), %d (day), ..., %z (zone OR UTC offset)
    a=datetime.strptime(t1,"%a %d %b %Y %H:%M:%S %z")
    # a => 2015-05-02 19:54:36+05:30
    # type(a) => <class 'datetime.datetime'>

    b=datetime.strptime(t2,"%a %d %b %Y %H:%M:%S %z")

    # a-b => 1 day, 0:30:00
    # type(a-b) => <class 'datetime.timedelta'>
    # (a-b).total_seconds() => 88200.0

    return str(int(abs((a-b).total_seconds())))
'''
given two timestamps in the following format: Day dd Mon yyyy hh:mm:ss +xxxx.
Here +xxxx represents the time zone. Your task is to print the absolute␣
 ↪difference (in seconds) between them.

1 => first line contains T, the number of testcases.
```

```
Sat 02 May 2015 19:54:36 +0530 => each testcase contains 2 lines, representing␣
 ↪time t1 and time t2.
Fri 01 May 2015 13:54:36 -0000
'''

for _ in range(int(input())):
    t1 = input()
    t2 = input()
delta = time_delta(t1, t2)
print(delta)
```

```
1
Sat 02 May 2015 19:54:36 +0530
Fri 01 May 2015 13:54:36 -0000
88200
```

[6]:
```
################################################################################
'''
1222311 => single line of input consisting of the string S.

(1, 1) (3, 2) (1, 3) (2, 1) => O/P
First, the character 1 occurs only once. It is replaced by (1,1). Then the␣
 ↪character 2 occurs three times, and it
is replaced by (3 => occurence count, 2 => character/element) and so on.
'''

# list(groupby(input())) => [('1', <itertools._grouper object at␣
 ↪0x7f330c055890>),
#      ('2', <itertools._grouper object at 0x7f330474a950>), ('3', <itertools.
 ↪_grouper object at 0x7f330474a590>),
#      ('1', <itertools._grouper object at 0x7f330474a510>)]


for k, c in groupby(input()):
    # k, c => 1, <itertools._grouper object at 0x7f33047a4250>
    # list(c) => 1
    # %d => this placeholder is used to represent an integer value.
    print("(%d, %d)" % ( len(list(c)), int(k) ), end=' ')
```

```
1222311
(1, 1) (3, 2) (1, 3) (2, 1)
```

[ ]:
```

```

[20]:
```
################################################################################
# taking input
n, m = input().split()
```

```
sc_ar = input().split()

# converting to set
A = set(input().split())
B = set(input().split())

'''
There is an array of n integers. There are also 2 disjoint sets, A and B, each␣
 ↪containing m integers.
You like all the integers in set A and dislike all the integers in set B. Your␣
 ↪initial happiness is 0.
For each i integer in the array , if i e A, you add 1 to your happiness. If i e␣
 ↪B, you add -1 to your happiness.
Otherwise, your happiness does not change. Output your final happiness at the␣
 ↪end.

3 2 => first line contains integers n and m separated by a space.
1 5 3 => second line contains n integers, the elements of the array.
3 1 =>  third and fourth lines contain m integers, A and B, respectively.
5 7
'''

# A => {'1', '3'}

# (i in A) => True
# (i in A) - (i in B) => 1
# (True-True) => 0
# (True-False) => 1

print (sum([(i in A) - (i in B) for i in sc_ar]))
```

```
3 2
1 5 3
3 1
5 7
1
```

[23]:
```
#####################################################################

a = int(input())
b = int(input())


M = math.sqrt(a**2 + b**2)
# acos => return the "arc cosine" (cosine inverse) of x, in radians.
theta = math.acos(b/M )
```

```
'''
ABC is a right triangle, 90 degree at B.
Point M is the midpoint of hypotenuse AC.
You are given the lengths AB and BC.
Your task is to find angle MBC in degrees.

10 => first line contains the length of side AB.
10 => second line contains the length of side BC.
'''
# theta => 0.7853981633974484
# math.degrees(theta) => 45.00000000000001
# int(round(math.degrees(theta),0)) => 45
print(int( round(math.degrees(theta),0) ),'\u00B0',sep='')
```

```
10
10
45°
```

[7]:
```
#########################################################################

#from collections import deque

'''
There is a horizontal row of n cubes. The length of each cube is given. You␣
 ↪need to create a new vertical pile
of cubes. The new pile should follow these directions:
cube[j]^cube[i]                (if cube[i] is on top of cube[j]) then
sideLength[j]>=sideLength[i].

When stacking the cubes, you can only pick up either the leftmost or the␣
 ↪rightmost cube each time.
After choosing the rightmost element, choose the leftmost element and␣
 ↪vice-versa.
Print "Yes" if it is possible to stack the cubes. Otherwise, print "No".
*Hint*: "Yes" is possible for the cases where innermost elements are less than␣
 ↪outermost elements.

1 => first line contains a single integer T, the number of test cases.
6 => first line of each test case contains n, the number of cubes.
4 3 2 1 3 4 => second line contains n space separated integers, denoting the␣
 ↪sideLengths of each cube.

1
3
1 3 2
'''
```

```
for i in range(int(input())):
    size = int(input())
    top = 2**31 # maximum value that an integer can take on
    # top => 2147483648

    d = deque(map(int,input().split()))
    # d => deque([4, 3, 2, 1, 3, 4])

    for j in range(len(d)):
        if d[0]>=d[len(d)-1] and d[0]<=top:
            top = d.popleft()
            # top => 4
            # d => deque([3, 2, 1, 3, 4])
        elif d[len(d)-1]<=top:
            top = d.pop()
            # top => 4
            # d => deque([3, 2, 1, 3])
        else:
            print('No')
            break
#         if j==1: break

    if len(d) == 0:
        print('Yes')
```

```
1
6
4 3 2 1 3 4
Yes
```

[49]:
```
##########################################################################
class Complex(object):
    def __init__(self, real, imaginary):
        self.real = real
        self.imaginary = imaginary

    def __add__(self, no):
        # no => 5.00+6.00i
        return Complex(self.real + no.real , self.imaginary + no.imaginary)

    def __sub__(self, no):
        return Complex(self.real - no.real , self.imaginary - no.imaginary)

    def __mul__(self, no):
        # complex(self.real , self.imaginary) => (2+1j)
```

```
        prod = complex(self.real , self.imaginary)*complex(no.real , no.
 ↪imaginary)
        return Complex(prod.real , prod.imag)

    def __truediv__(self, no):
        div = complex(self.real , self.imaginary)/complex(no.real , no.
 ↪imaginary)
        return Complex(div.real , div.imag)

    def mod(self):
        m = math.sqrt(self.real**2 + self.imaginary**2)
        return Complex(m,0)

    def __str__(self):
        if self.imaginary >= 0:
            result = "%.2f+%.2fi" % (self.real, self.imaginary)
        else:
            result = "%.2f-%.2fi" % (self.real, abs(self.imaginary))
        return result



c = map(float, input().split())
d = map(float, input().split())

# *c => 2.0 1.0

x = Complex(*c)
# x => 2.00+1.00i
y = Complex(*d)
# y => 5.00+6.00i

'''
given two complex numbers (C and D), and you have to print the result of their␣
 ↪addition, subtraction,
multiplication, division and modulus operations i.e., C+D, C-D, C*D, C/D,␣
 ↪mod(C) and mod(D).

2 1 => two lines of input for 2 numbers: the real and imaginary part of a each␣
 ↪number.
5 6
'''



print(*map(str, [x+y, x-y, x*y, x/y, x.mod(), y.mod()]), sep='\n')
```

```
2 1
```

```
5 6
(2+1j)
7.00+7.00i
-3.00-5.00i
4.00+17.00i
0.26-0.11i
2.24+0.00i
7.81+0.00i
```

[53]:
```
print(complex(5,10))
complex(5,10).imag
```

```
(5+10j)
```

[53]:
```
10.0
```

[70]:
```
##########################################################################

n = int(input())
ls = input().split()
# ls => ['a', 'a', 'c', 'd']
k = int(input())

# .combinations(iterable, r) => return r length subsequences of elements from␣
 ↪the input iterable.

# combinations(ls, k) => <itertools.combinations object at 0x7f32eafaed70>
# combinations(ls, k) => generate all possible combinations of k letters from␣
 ↪the input list of letters.
com = list(combinations(ls, k))
# com => [('a', 'a'), ('a', 'c'), ('a', 'd'), ('a', 'c'), ('a', 'd'), ('c',␣
 ↪'d')]
tol = [i for i in com  if "a" in i]

# tol => [('a', 'a'), ('a', 'c'), ('a', 'd'), ('a', 'c'), ('a', 'd')]


'''
You are given a list of N lowercase English letters. For a given integer K, you␣
 ↪can select any K indices
(assume 1-based indexing) with a uniform probability from the list. Find the␣
 ↪probability that at least one of
the K indices selected will contain the letter: 'a'.

4 => first line contains the integer N, denoting the length of the list.
a a c d => next line consists of N space-separated lowercase English letters,␣
 ↪denoting the elements of the list.
```

```
2 => third and the last line of input contains the integer K, denoting the␣
 ↪number of indices to be selected.
'''
print(f'{(len(tol)/len(com)):.12f}')
```

```
4
a a c d
2
0.83
```

[ ]:

[86]:
```
###############################################################################

'''
given a positive integer N. print a numerical triangle of height N-1 like the␣
 ↪one below:
1
22
333
4444

5 => a single line containing integer, N.
'''

for i in range(1,int(input())):
    # bin(2**i - 1) => 0b1           (for i=1, 2**i =2)
    # bin(2**i - 1) => 0b11          (for i=2, 2**i =4)
    # bin(2**i - 1) => 0b111         (for i=3, 2**3 =8)

    # binary of "1, 3, 7, 15, 31, ..." is "1, 11, 111, 1111, 11111, ..."
    print (i * int(bin(2**i - 1)[2:]))
```

```
5
1
22
333
4444
```

[ ]:

[91]:
```
###############################################################################
'''
print a palindromic triangle of size N.
a palindromic triangle of size 5 is:
1
121
```

```
12321
1234321
123454321

5 => a single line of input containing the integer N.
'''
for i in range(1,int(input())+1):
    # (10**i - 1) => 9      (for i=1)
    # (10**i - 1) => 99     (for i=2)

    # power of "1, 11 , 111, 111, ..." is 2 the o/p is "1, 121, 12321, 1234321,
 ↪...."
    print ( ((10**i - 1)//9)**2 )
```

```
5
1
121
12321
1234321
123454321
```

[111]:
```
##########################################################################################
A = [[32, 9, 26, 57, 5],
[32, 39, 89, 96, 1],
[84, 61, 56, 99, 84],
[55, 13, 14, 46, 60],
[6, 70, 27, 7, 32]]

B = [[35, 94, 4, 62, 67],
[97, 81, 26, 21, 79],
[56, 63, 35, 57, 10],
[22, 18, 16, 88, 43],
[67, 87, 82, 16, 22]]

print(np.array(A).shape, np.array(B).shape)
'''
Matrix multiplication of A and B.
'''
# A*B
#print(np.dot(A,B)) # OR
#print(np.matmul(A,B)) # preferred method # OR
np.array(A) @ np.array(B)
```

```
(5, 5) (5, 5)
```

[111]:
```
array([[ 5038,  6836,  2594,  8751,  5676],
       [12066, 13589,  5875, 16340, 10265],
```

```
        [19799, 25455, 12354, 19737, 17112],
        [ 9002, 13153,  6704,  9489,  8150],
        [10810, 10845,  5525,  4509,  7207]])
```

[114]:
```
A = [32, 9, 26, 57, 5]
B = [35, 97, 56, 22, 67]
#print(np.dot(A,B)) # OR
print(np.matmul(A,B)) # preferred method
```

```
5038
```

[159]:
```
#########################################################################
'''
list all primes number below and equal to n.
'''
def era1():
    n = int(input("Introduce a number: ").strip())
    A = range(2, n+1)
    # A => range(2, 31)
    B, C = [], A
    # math.sqrt(n) => 5.477225575051661
    while C[0]< math.sqrt(n): #Condition
        firstElement = C[0]

        B += [firstElement]

        C = [x for x in C if x%firstElement!=0]
        # C => [3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]       (1st␣
 ↪entry in loop)
        # C => [5, 7, 11, 13, 17, 19, 23, 25, 29]                         (2nd␣
 ↪entry in loop)
        # C => [7, 11, 13, 17, 19, 23, 29]                                (3rd␣
 ↪entry in loop)

    return B+C

era1()
#print(era1())
```

```
Introduce a number: 30
```

[159]:
```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

[ ]:
```
#########################################################################
```

[166]:
```
%%time
# prime factors of a number.
```

```
def prime_factors(n):
    i = 2
    factors = []
    while i * i <= n:

        if n % i:
            i += 1
        else:
            n //= i
            factors.append(i)
    if n > 1:
        factors.append(n)
    return factors

print(prime_factors(126))
```

```
[2, 3, 3, 7]
CPU times: user 47 µs, sys: 19 µs, total: 66 µs
Wall time: 62.7 µs
```

[ ]:

[2]:
```
##############################################################################
def is_leap(year):
    leap = False

    if ((year % 4 ==0) and (year % 100 != 0)) | ((year % 400 == 0) and (year %␣
 ↪100 == 0)):
        leap = True

    return leap
'''
In the Gregorian calendar, identify leap years as:
Evenly divisible by 4 and not 100.
OR
Evenly divisible by 400 and 100.
'''

is_leap(2017), is_leap(2016)
```

[2]: (False, True)

[10]:
```
##############################################################################

row, col = 9, 27
wel = "WELCOME"
```

```
# .center(width[, fillchar]) => return centered in a string of length width. padding is done using the specified
#         - fillchar (default is an ASCII space).
print(wel.center(col,"-"))
```

```
----------WELCOME----------
```

[18]:
```
################################################################################
i = 54897
w = [int(x) for x in str(i)]
w
```

[18]:
```
[5, 4, 8, 9, 7]
```

[20]:
```
l = ['first']
while len(l):
    print('hello')
    l=[]
```

```
hello
```

[21]:
```
l=[5,4,3]
while 2 not in l:
    print('hello')
    l.append(2)
```

```
hello
```

[24]:
```
if not (6 in l): # ! instead of not gives SyntaxError.
    print('hello')
```

```
hello
```

[ ]:
```

```

[44]:
```
'''
factorial of a number
'''
def factorial_(n):
    if n == 0:
        return 1
    elif n == 1:
        return 1
    else:
        return n*factorial_(n-1)

factorial_(6)
```

[44]: 720

[ ]: