PYTHON MEDIUM HACKERANK SOLVED PROBLEMS

**1.Merge the Tools!**

**def** merge\_the\_tools(string, k):

*# your code goes here*

    n = **len**(string)

    s\_split = ["".join(string[0+i: k+i]) **for** i **in** **range**(0, n, k)]

**for** part **in** s\_split:

**print**("".join(**set**(part)))

**if** \_\_name\_\_ == '\_\_main\_\_':

    string, k = **input**(), **int**(**input**())

    merge\_the\_tools(string, k)

# 2.Write a function

**def** is\_leap(year):

    leap = **False**

*# Write your logic here*

**return** leap

year = **int**(**input**())

# 3.Time Delta

*#!/bin/python3*

**import** math

**import** os

**import** random

**import** re

**import** sys

*# Complete the time\_delta function below.*

**import** datetime

**def** time\_delta(t1, t2):

    dtfmt = '%a %d %b %Y %H:%M:%S %z'

    dt1 = datetime.datetime.strptime(t1, dtfmt)

    dt2 = datetime.datetime.strptime(t2, dtfmt)

    delta = **abs**(dt1 - dt2)

**return** '%d' % **round**(delta.total\_seconds())

**if** \_\_name\_\_ == '\_\_main\_\_':

    fptr = **open**(os.environ['OUTPUT\_PATH'], 'w')

    t = **int**(**input**())

**for** t\_itr **in** **range**(t):

        t1 = **input**()

        t2 = **input**()

        delta = time\_delta(t1, t2)

        fptr.write(delta + '\n')

    fptr.close()

# 4.Find Angle MBC

*# Enter your code here. Read input from STDIN. Print output to STDOUT*

**import** math **as** m

ab=**int**(**input**())

bc=**int**(**input**())

ac=m.sqrt(ab\*\*2+bc\*\*2)

cm=ac/2

bm=m.sqrt(cm\*\*2+bc\*\*2-2\*bc\*cm\*bc/ac)

theta=m.degrees(m.asin(ab\*cm/(ac\*bm)))

**print**(**round**(theta),**chr**(176),sep='')

# 5.No Idea!

*# Enter your code here. Read input from STDIN. Print output to STDOUT*

**from** collections **import** Counter

**from** operator **import** itemgetter

n, m = **map**(**int**, **input**().split())

arr = **map**(**int**, **input**().split())

A = **map**(**int**, **input**().split())

B = **map**(**int**, **input**().split())

counter = Counter(arr)

happiness = **sum**(itemgetter(\*A)(counter))

sadness = **sum**(itemgetter(\*B)(counter))

**print**(happiness - sadness)

# 6.Word Order

*# Enter your code here. Read input from STDIN. Print output to STDOUT*

n = **int**(**input**())

words = **dict**()

**for** i **in** **range**(n):

    word = **input**()

    words[word] = words.get(word, 0) + 1

**print**(**len**(words))

**print**(\*words.values())

# 7.The Minion Game

**def** minion\_game(string):

*# your code goes here*

    vowels = 'AEIOU'

    kevin\_score, stuart\_score = 0, 0

    strlen = **len**(string)

**for** i **in** **range**(0, strlen):

**if** string[i] **in** vowels:

            kevin\_score += strlen-i

**else**:

            stuart\_score += strlen-i

**if** kevin\_score > stuart\_score:

**print**(f"Kevin {kevin\_score}")

**elif** kevin\_score < stuart\_score:

**print**(f"Stuart {stuart\_score}")

**else**:

**print**("Draw")

**if** \_\_name\_\_ == '\_\_main\_\_':

# 8.Compress the String!

*# Enter your code here. Read input from STDIN. Print output to STDOUT*

**from** itertools **import** groupby

s=[]

**for** i **in** ([**list**(r) **for** j, r **in** groupby(**input**())]):

   s.append((**len**(i),**int**(i[0])))

**print**(\*s)

# 9.Piling Up!

*# Enter your code here. Read input from STDIN. Print output to STDOUT*

T = **int**(**input**())

tests = []

**for** \_ **in** **range**(T):

**input**()

    tests.append(**list**(**map**(**int**, **input**().split())))

**def** getMost(arr):

    left = arr[0]

    right = arr[-1]

**if** right >= left:

        arr.pop()

**return** right

**else**:

        arr.pop(0)

**return** left

**def** isStacked(blocks):

    picked = []

    picked.append(getMost(blocks))

    i = 0

**while** **len**(blocks) != 0:

        picked.append(getMost(blocks))

**if** picked[i] < picked[i+1]:

**return** "No"

        i+=1

**return** "Yes"

**for** test **in** tests:

**print**(isStacked(test))

# 10.Triangle Quest 2

**for** i **in** **range**(1,**int**(**input**())+1): *#More than 2 lines will result in 0 score. Do not leave a blank line also*

**print**(**pow**((**pow**(10,i)//9),2))

# 11.Iterables and Iterators

*# Enter your code here. Read input from STDIN. Print output to STDOUT*

**from** itertools **import** combinations

N, letters, K = **int**(**input**()), **input**().split(), **int**(**input**())

a\_idx = **set**(idx **for** idx **in** **range**(N) **if** letters[idx] == 'a')

total, witha = 0, 0

**for** combination **in** combinations(**range**(N), K):

    total += 1

**if** a\_idx.intersection(combination):

        witha += 1

**print**(**round**(witha/total, 3))

# 12.Triangle Quest

**for** num **in** **range**(1, **int**(**input**())):

**print**((10\*\*num//9)\*num)

# 13.Classes: Dealing with Complex Numbers

**import** math

**class** Complex:

**def** \_\_init\_\_(**self**, real, img):

**self**.r = real

**self**.i = img

**def** \_\_str\_\_(**self**):

**return** f'{self.r:.2f}{self.i:+.2f}i'

**def** \_\_add\_\_(**self**, other):

**return** Complex(**self**.r + other.r, **self**.i + other.i)

**def** \_\_sub\_\_(**self**, other):

**return** Complex(**self**.r - other.r, **self**.i - other.i)

**def** \_\_mul\_\_(**self**, other):

        real = **self**.r \* other.r - **self**.i \* other.i

        img = **self**.r \* other.i + **self**.i \* other.r

**return** Complex(real, img)

**def** \_\_truediv\_\_(**self**, other):

        a, b = **self**, other

        real = (a.r\*b.r + a.i\*b.i) / (b.r\*\*2 + b.i\*\*2)

        img = (b.r\*a.i - a.r\*b.i) / (b.r\*\*2 + b.i\*\*2)

**return** Complex(real, img)

**def** mod(**self**):

*# by the way modulus is not a complex num*

**return** Complex(math.sqrt(**self**.r \*\* 2 + **self**.i \*\* 2), 0)

**if** \_\_name\_\_ == '\_\_main\_\_':

# 14.Athlete Sort

*#!/bin/python3*

**import** math

**import** os

**import** random

**import** re

**import** sys

**if** \_\_name\_\_ == '\_\_main\_\_':

    nm = **input**().split()

    n = **int**(nm[0])

    m = **int**(nm[1])

    arr = []

**for** \_ **in** **range**(n):

        arr.append(**list**(**map**(**int**, **input**().rstrip().split())))

    k = **int**(**input**())

**for** r **in** **sorted**(arr, key=**lambda** x: (x[k])):

**print**(\*r)

# 15.ginortS

*# Enter your code here. Read input from STDIN. Print output to STDOUT*

**def** key\_func(char: **str**) -> **int**:

**if** **not** char.isalnum():

**return** char

**if** char.islower():

**return** **ord**(char)-**ord**('a')

**if** char.isupper():

**return** **ord**(char)-**ord**('A') + 26

**if** **int**(char) % 2 != 0:

**return** **int**(char)//2 + 52

**return** **int**(char)//2 + 57

**print**(''.join(**sorted**(**input**(), key=key\_func)))

# Validating Email Addresses With a Filter

**def** fun(s):

*# return True if s is a valid email, else return Falsedef fun(s):*

**import** re

**return** **bool**(re.match(r'^[\w-]+@[A-Za-z\d]+\.[a-zA-Z]{1,3}$', s))

**def** filter\_mail(emails):

# 16.Reduce Function

**from** fractions **import** Fraction

**from** functools **import** **reduce**

**def** product(fracs):

    t = **reduce**(**lambda** x, y : x \* y,[\*fracs])*# complete this line with a reduce statement*

**return** t.numerator, t.denominator

**if** \_\_name\_\_ == '\_\_main\_\_':

    fracs = []

**for** \_ **in** **range**(**int**(**input**())):

        fracs.append(Fraction(\***map**(**int**, **input**().split())))

    result = product(fracs)

**print**(\*result)

# 17.Regex Substitution

*# Enter your code here. Read input from STDIN. Print output to STDOUT*

**from** re **import** **compile**

**def** getfunc\_sub():

    translate = {'&&': 'and', '||': 'or'}

**def** sub\_andor(m):

**return** translate[m.group('andor')]

**return** sub\_andor

regex = **compile**(r'(?<=\s)(?P<andor>[&][&]|[|][|])(?=\s)')

**for** \_ **in** **range**(**int**(**input**())):

**print**(regex.sub(getfunc\_sub(), **input**()))

# 18. Validating Credit Card Numbers

*# Enter your code here. Read input from STDIN. Print output to STDOUT*

**import** re

**from** itertools **import** groupby

**def** is\_valid(s):

**for** key, group **in** groupby(re.sub(r'[^\d]', '', s)):

**if** **len**(**list**(group)) > 3:

**return** 'Invalid'

**return** 'Valid' **if** re.match(r'^[456]\d{3}-?\d{4}-?\d{4}-?\d{4}$', s) **else** 'Invalid'

[**print**(is\_valid(**input**())) **for** \_ **in** **range**(**int**(**input**()))]

# 19. Words Score

**def** is\_vowel(letter):

**return** letter **in** ['a', 'e', 'i', 'o', 'u', 'y']

**def** score\_words(words):

    score = 0

**for** word **in** words:

        num\_vowels = 0

**for** letter **in** word:

**if** is\_vowel(letter):

                num\_vowels += 1

**if** num\_vowels % 2 == 0:

            score += 2

**elif** num\_vowels % 2 != 0:

            score += 1

**return** score

n = **int**(**input**())

words = **input**().split()

**print**(score\_words(words))

# 20. Default Arguments

**class** EvenStream(**object**):

**def** \_\_init\_\_(**self**):

**self**.current = 0

**def** get\_next(**self**):

        to\_return = **self**.current

**self**.current += 2

**return** to\_return

**class** OddStream(**object**):

**def** \_\_init\_\_(**self**):

**self**.current = 1

**def** get\_next(**self**):

        to\_return = **self**.current

**self**.current += 2

**return** to\_return

**class** EvenStream(**object**):

**def** \_\_init\_\_(**self**):

**self**.current = 0

**def** get\_next(**self**):

        to\_return = **self**.current

**self**.current += 2

**return** to\_return

**class** OddStream(**object**):

**def** \_\_init\_\_(**self**):

**self**.current = 1

**def** get\_next(**self**):

        to\_return = **self**.current

**self**.current += 2

**return** to\_return

**def** print\_from\_stream(n, stream=**None**):

**if** stream==**None**:

        stream = EvenStream()

**for** \_ **in** **range**(n):

**print**(stream.get\_next())

queries = **int**(**input**())

**for** \_ **in** **range**(queries):

    stream\_name, n = **input**().split()

    n = **int**(n)

**if** stream\_name == "even":

        print\_from\_stream(n)

**else**:

        print\_from\_stream(n, OddStream())

# 21. Company Logo

*#!/bin/python3*

**import** math

**import** os

**import** random

**import** re

**import** sys

**if** \_\_name\_\_ == '\_\_main\_\_':

    s = **input**()

    result = {}

    count = 0

**for** l **in** s:

**if** l **not** **in** result.keys():

            result[l] = 0

**if** l **in** result.keys():

            result[l] += 1

    result = **sorted**(result.items(), key = **lambda** x: (-x[1], x[0]))

**for** i **in** **range**(3):

**print**(result[i][0], result[i][1])