COURSE: ARTIFICIAL INTELLIGENCE

ASSIGNMENT NO: 03

Submitted By

Muhammad Riaz

Reg Number

452759

Submitted To

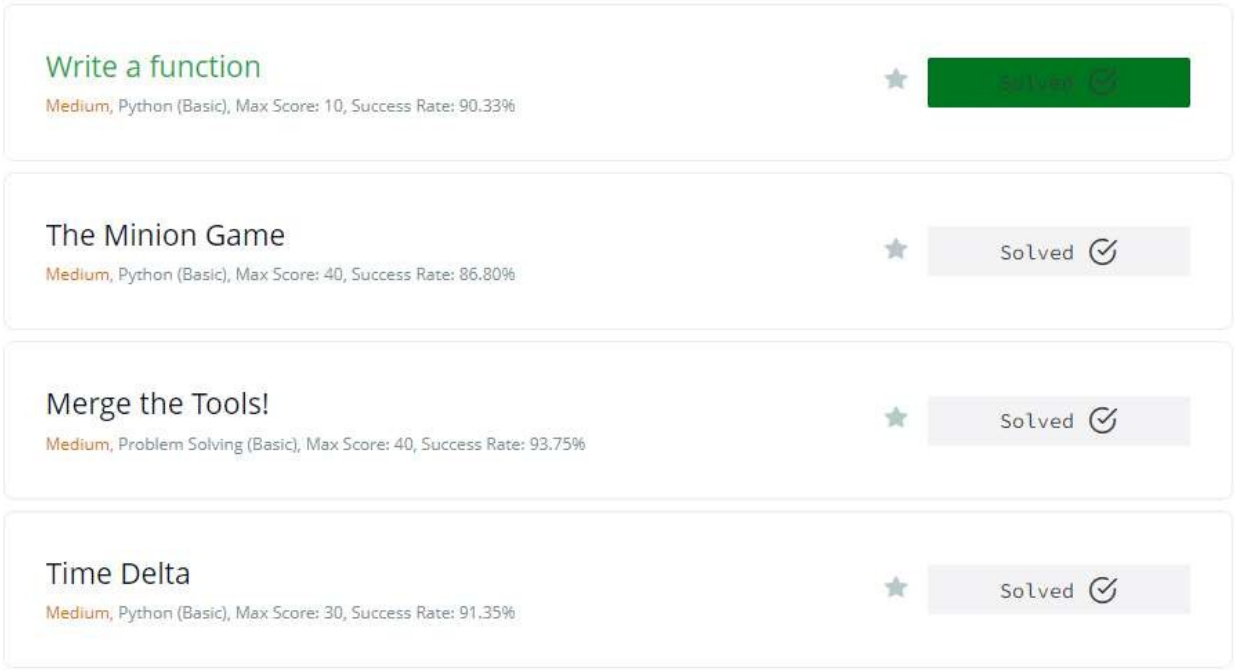
Dr Yasar Ayaz

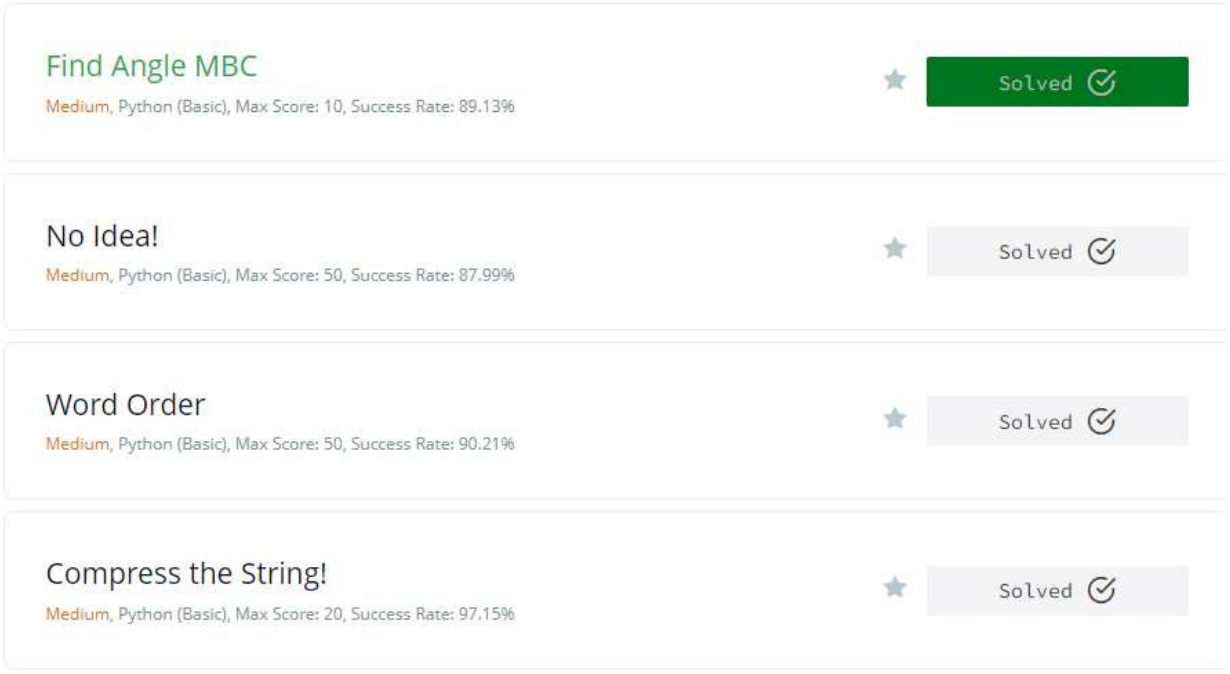
Department of Robotics and Artificial Intelligence

SCHOOL OF MECHANICAL & MANUFACTURING ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLO

MEDIUM CHALLENGES SUMMARY :





Company Logo
Medium, Problem Solving (Basic), Max Score: 30, Success Rate: 89.83%
Solved

Piling Up!
Medium, Python (Basic), Max Score: 50, Success Rate: 90.63%
Solved

Triangle Quest 2
Medium, Python (Basic), Max Score: 20, Success Rate: 95.39%
Solved

Iterables and Iterators
Medium, Python (Basic), Max Score: 40, Success Rate: 96.60%
Solved

Triangle Quest
Medium, Python (Basic), Max Score: 20, Success Rate: 93.84%
Solved

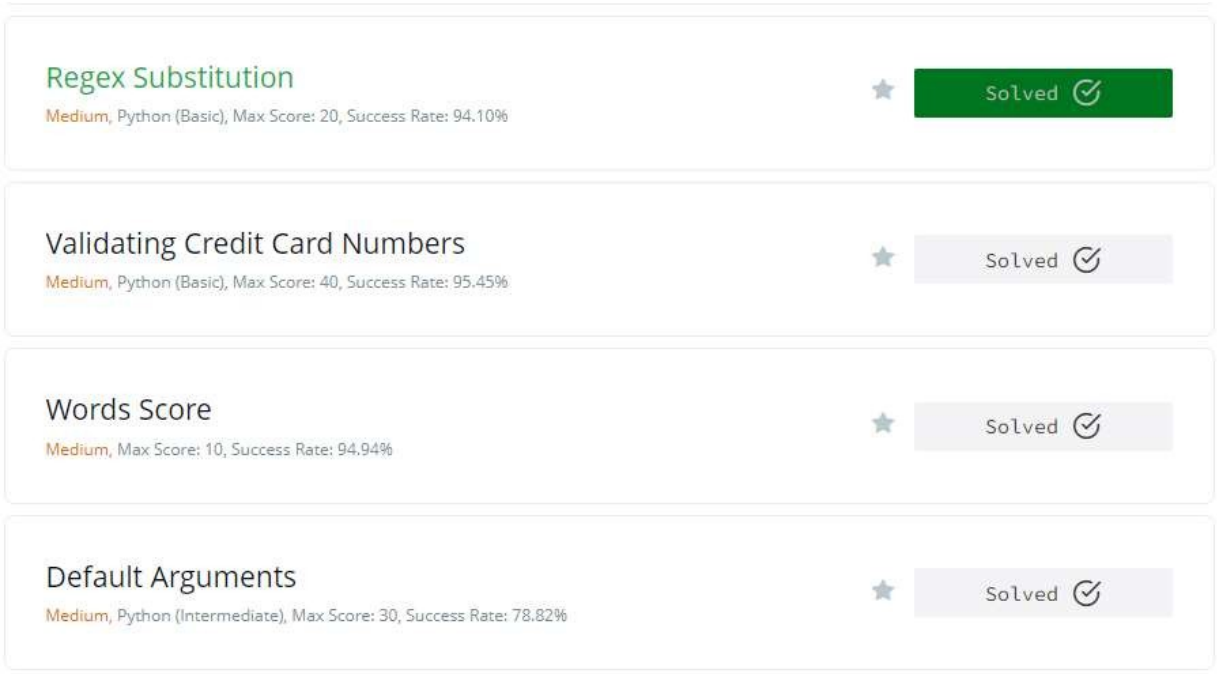
Classes: Dealing with Complex Numbers
Medium, Python (Basic), Max Score: 20, Success Rate: 90.91%
Solved

Athlete Sort
Medium, Python (Basic), Max Score: 30, Success Rate: 95.52%
Solved

ginortS
Medium, Python (Basic), Max Score: 40, Success Rate: 97.63%
Solved

Validating Email Addresses With a Filter
Medium, Python (Basic), Max Score: 20, Success Rate: 90.82%
Solved

Reduce Function
Medium, Max Score: 30, Success Rate: 98.38%
Solved



TASKS :

TASK 1 :

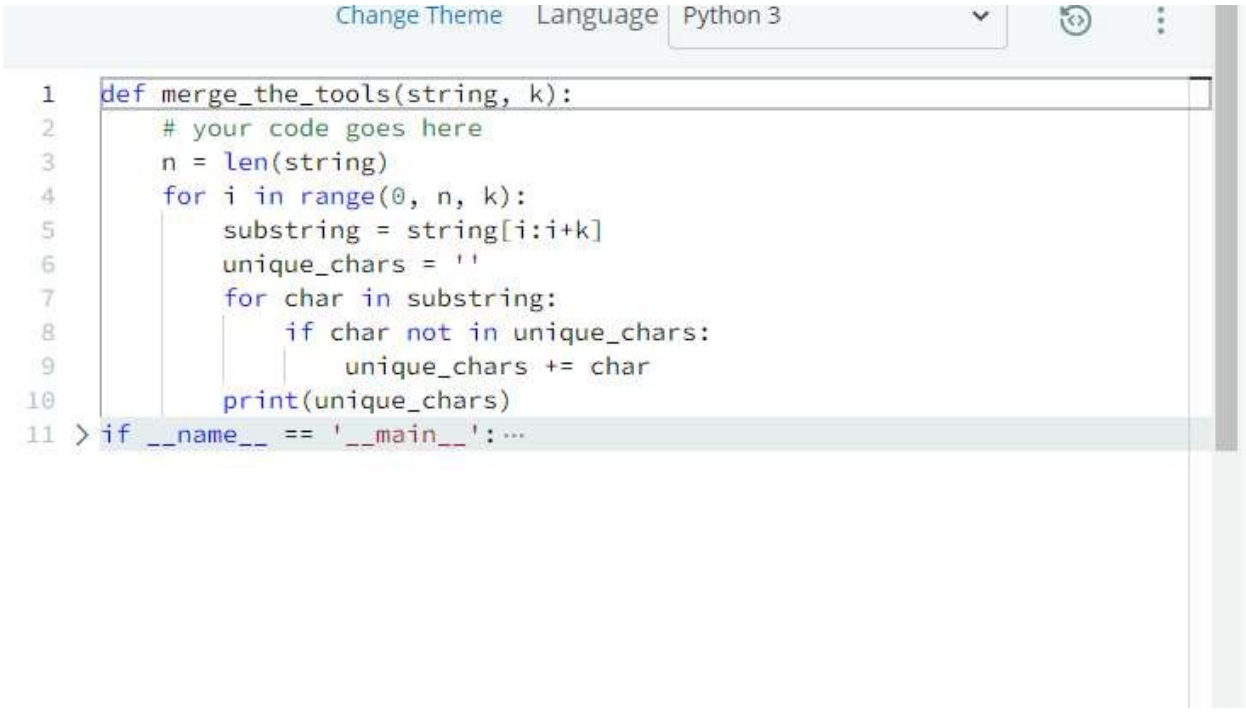
```
def is_leap(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False

year = int(input())
```

TASK 2 :

```python
def minion_game(string):
    # your code goes here
    vowels = "AEIOU"
    stuart_score = 0
    kevin_score = 0
    length = len(string)

    for i in range(length):
        if string[i] in vowels:
            kevin_score += length - i
        else:
            stuart_score += length - i

    if stuart_score > kevin_score:
        print("Stuart", stuart_score)
    elif kevin_score > stuart_score:
        print("Kevin", kevin_score)
    else:
        print("Draw")
```

Line: 1 Col: 1

TASK 3 :




```python
def merge_the_tools(string, k):
    # your code goes here
    n = len(string)
    for i in range(0, n, k):
        substring = string[i:i+k]
        unique_chars = ''
        for char in substring:
            if char not in unique_chars:
                unique_chars += char
        print(unique_chars)
if __name__ == '__main__':...
```

TASK 4 :

```python
import math
```

```
import os
import random
import re
import sys

# Complete the time_delta function below.
def time_delta(t1, t2):

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()
```

TASK 5:

```
import math

def find_angle(opposite, adjacent):
    # Calculate the angle in radians using arctan
    angle_radians = math.atan(opposite / adjacent)

    # Convert the angle from radians to degrees
    angle_degrees = math.degrees(angle_radians)

    return round(angle_degrees)  # Round to the nearest integer

# Read input lengths
side_a = float(input())
side_b = float(input())

# Calculate and print the angle
degree_symbol = chr(176)
result = find_angle(side_a, side_b)
print(f"{result}{degree_symbol}")
```

TASK 6:

```
if __name__ == '__main__':
    n, m = map(int, input().split())
    arr = list(map(int, input().split()))
    set_a = set(map(int, input().split()))
    set_b = set(map(int, input().split()))

    happiness = 0

    for num in arr:
        if num in set_a:
            happiness += 1
        elif num in set_b:
            happiness -= 1

    print(happiness)
```

TASK 7:

```
rom collections import OrderedDict

def word_count(words):
    word_dict = OrderedDict()

    for word in words:
        if word in word_dict:
            word_dict[word] += 1
        else:
            word_dict[word] = 1

    return word_dict

if __name__ == "__main__":
    n = int(input().strip())
    words = [input().strip() for _ in range(n)]

    word_dict = word_count(words)

    print(len(word_dict))
    print(" ".join(map(str, word_dict.values())))
```

TASK 8:

```
from itertools import groupby
```

```
def compress_string(s):
    # Use groupby to group consecutive characters
    groups = [(len(list(g)), int(k)) for k, g in groupby(s)]

    # Format the result as specified
    result = " ".join(f"({count}, {char})" for count, char in group
s)

    return result

# Read input string
input_string = input()

# Compress the string and print the result
output_string = compress_string(input_string)
print(output_string)
```

TASK 9:

```
import math
import os
import random
import re
import sys



from collections import Counter

def print_top_characters(s):
    counter = Counter(s)
    sorted_characters = sorted(counter.items(), key=lambda x: (-
x[1], x[0]))

    for char, count in sorted_characters[:3]:
        print(char, count)

if __name__ == '__main__':
    s = input().strip()
    print_top_characters(s)
```

TASK 10:

```
from collections import deque

def can_stack_cubes(t, test_cases):
    for test in test_cases:
        n, cubes = test
        stack = deque(cubes)
        last_cube = float('inf')
```

```
        while stack:
            left, right = stack[0], stack[-1]

            if left >= right and left <= last_cube:
                last_cube = stack.popleft()
            elif right >= left and right <= last_cube:
                last_cube = stack.pop()
            else:
                print("No")
                break
        else:
            print("Yes")

if __name__ == '__main__':
    t = int(input().strip())
    test_cases = []

    for _ in range(t):
        n = int(input().strip())
        cubes = list(map(int, input().split()))
        test_cases.append((n, cubes))

    can_stack_cubes(t, test_cases)
```

TASK 11:

```
for i in range(1, int(input()) + 1):
    print(((10 ** i - 1) // 9) ** 2
```

TASK 12:

```
from itertools import combinations

def probability_of_a(n, letters, k):
    indices_without_a = [i for i, letter in enumerate(letters, star
t=1) if letter != 'a']
    total_indices = n
    indices_to_select = k

    probability_none_contain_a = len(list(combinations(indices_with
out_a, indices_to_select))) / \
                                 len(list(combinations(range(1, tot
al_indices + 1), indices_to_select)))

    probability_at_least_one_contains_a = 1 - probability_none_cont
ain_a

    return round(probability_at_least_one_contains_a, 4)
```

```
if __name__ == "__main__":
    n = int(input())
    letters = input().split()
    k = int(input())

    result = probability_of_a(n, letters, k)
    print(result)
```

TASK 13 :

```
for i in range(1, int(input()) ):
    print((10 ** i - 1) // 9 * i)
```

TASK 15:

```
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())
    [print(*i) for i in sorted(arr, key=lambda attribute: attribute
[k])]
```

TASK 14:

```
import math


class Complex(object):
    def __init__(self, real, imaginary):
        self.real = real
```

```
        self.imaginary = imaginary

    def__add__(self, no):
        return Complex(self.real + no.real, self.imaginary + no.ima
ginary)

    def__sub__(self, no):
        return Complex(self.real - no.real, self.imaginary - no.ima
ginary)

    def__mul__(self, no):
        real_part = self.real * no.real - self.imaginary * no.imagi
nary
        imag_part = self.real * no.imaginary + self.imaginary * no.
real
        return Complex(real_part, imag_part)

    def__truediv__(self, no):
        den = no.real**2 + no.imaginary**2
        real_part = (self.real * no.real + self.imaginary * no.imag
inary) / den
        imag_part = (self.imaginary * no.real - self.real * no.imag
inary) / den
        return Complex(real_part, imag_part)

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

    def__str__(self):
        if self.imaginary == 0:
            result = "%.2f+0.00i" % (self.real)
        elif self.real == 0:
            if self.imaginary >= 0:
                result = "0.00+%.2fi" % (self.imaginary)
            else:
                result = "0.00-%.2fi" % (abs(self.imaginary))
        elif self.imaginary > 0:
            result = "%.2f+%.2fi" % (self.real, self.imaginary)
        else:
            result = "%.2f-
%.2fi" % (self.real, abs(self.imaginary))
        return result



if __name__ == '__main__':
    c = map(float, input().split())
    d = map(float, input().split())
```

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

TASK 16:

```
def custom_sort(c):
    if c.islower():
        return (0, c)
    elif c.isupper():
        return (1, c)
    elif c.isdigit():
        if int(c) % 2 == 1:
            return (2, c)
        else:
            return (3, c)

# Read input string
input_string = input()

# Sort the string using the custom sorting key
sorted_string = ''.join(sorted(input_string, key=custom_sort))

# Print the sorted string
print(sorted_string)
```

TASK 17:

```
import re
def fun(s):
# Define the conditions for a valid email
    pattern = re.compile(r'^[\w-]+@[A-Za-z0-9]+\.[A-Za-z]{1,3}$')
    return bool(re.match(pattern, s))


def filter_mail(emails):
    return list(filter(fun, emails))

if __name__ == '__main__':
    n = int(input())
    emails = []
    for _ in range(n):
        emails.append(input())

filtered_emails = filter_mail(emails)
filtered_emails.sort()
```

```
print(filtered_emails)
```

TASK 18:

```
from fractions import Fraction
from functools import reduce

def product(fracs):

    # Use reduce to find the product of all fractions
    result_fraction = reduce(lambda x, y: x * y, fracs)
    return result_fraction.numerator, result_fraction.denominator

if __name__ == '__main__':
    fracs = []
    for _ in range(int(input())):
        fracs.append(Fraction(*map(int, input().split())))
    result = product(fracs)
    print(*result)
```

TASK 19:

```
import re

def modify_symbols(text):
    # Replace '&&' with 'and' and '||' with 'or' with space on both
 sides
    modified_text = re.sub(r'(?<= )&&(?= )', 'and', text)
    modified_text = re.sub(r'(?<= )\|\|(?= )', 'or', modified_text)
    return modified_text

if __name__ == '__main__':
    # Read the number of lines
    n = int(input().strip())

    # Read and modify each line
    for _ in range(n):
        line =  input().strip()
        modified_line = modify_symbols(line)
        print(modified_line)
```

TASK 20:

```
import re

def is_valid_credit_card(card_number):
```

```
    # Check if the card number matches the specified pattern
    pattern = re.compile(r'^[4-6]\d{3}-?\d{4}-?\d{4}-?\d{4}$')
    if not re.match(pattern, card_number):
        return "Invalid"

    # Remove hyphens from the card number
    card_number = card_number.replace('-', '')

    # Check for consecutive repeated digits
    consecutive_repeated_digits = re.compile(r'(\d)(\1{3,})')
    if re.search(consecutive_repeated_digits, card_number):
        return "Invalid"

    return "Valid"

if __name__ == "__main__":
    n = int(input())

    for _ in range(n):
        card_number = input().strip()
        result = is_valid_credit_card(card_number)
        print(result)
```

TASK 21:

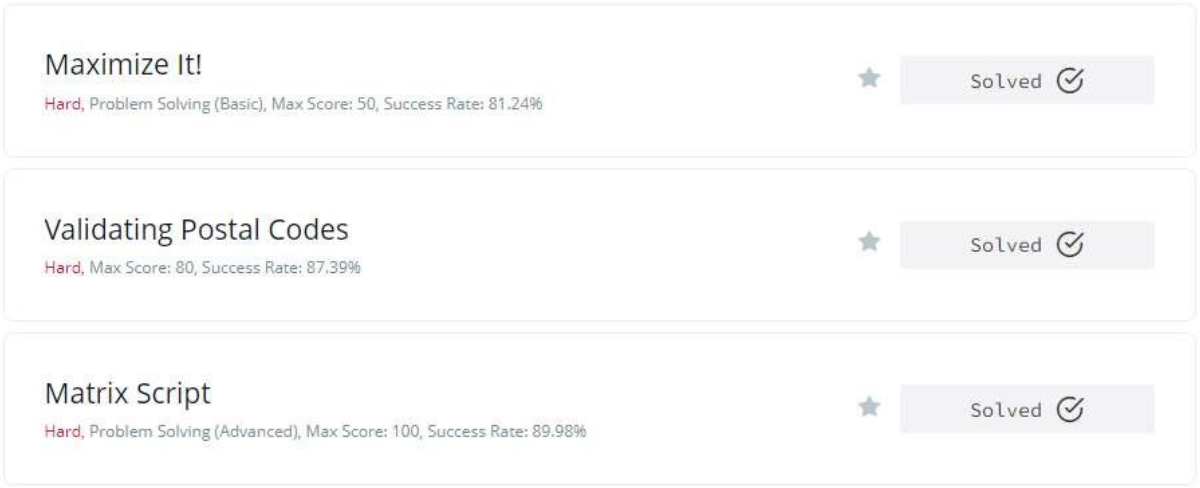
```
class EvenStream(object):
    def__init__(self):
        self.current = 0

    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 is 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())
```

TASK 22:

```
#!/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())
    [print(*i) for i in sorted(arr, key=lambda attribute: attribute
[k])]
```

HARD CHALLENGES



CODES OF THE TASKS

TASK 1 : MAXIMIZE IT :

```
from itertools import product

def maximize_value(n, m, lists):
    # Generate all possible combinations
    combinations = list(product(*lists))

    # Calculate the value for each combination and find the maximum
    max_value = max(sum(x**2 for x in combo) % m for combo in combi
nations)

    return max_value

# Input
n, m = map(int, input().split())
lists = [list(map(int, input().split()[1:])) for _ in range(n)]

# Output
result = maximize_value(n, m, lists)
print(result)
```

TASK 2 : Validating Postal Codes :

```
import re
regex_integer_in_range = r'^[1-9][0-9]{5}$'
regex_alternating_repetitive_digit_pair = r'(\d)(?=\d\1)'
```

```
import re
P = input()

print (bool(re.match(regex_integer_in_range, P))
and len(re.findall(regex_alternating_repetitive_digit_pair, P)) < 2
)
```

TASK 3 : MATRIX SCRIPT :

```
import re

def decode_matrix_script(rows, columns, matrix):
    # Transpose the matrix to read columns instead of rows
    transposed_matrix = list(map(list, zip(*matrix)))

    # Concatenate alphanumeric characters from each column
    decoded_script = ''.join([''.join(col) for col in transposed_ma
trix])

    # Replace symbols or spaces between alphanumeric characters wit
h a single space
    decoded_script = re.sub(r'(?<=[a-zA-Z0-9])[^a-zA-Z0-9]+(?=[a-
zA-Z0-9])', ' ', decoded_script)

    return decoded_script

# Input
rows, columns = map(int, input().split())
matrix = [input() for _ in range(rows)]

# Output
result = decode_matrix_script(rows, columns, matrix)
print(result)
```