

Algorithmic Methods of Data Mining

Claudio Giannini (2093898)

Fall 2023

Contents

0.1	Introduction	4
0.1.1	Exercise: "Say 'Hello, World!' With Python"	4
0.1.2	Exercise: "Python If-Else"	4
0.1.3	Exercise: "Arithmetic Operators"	4
0.1.4	Exercise: "Python: Division"	4
0.1.5	Exercise: "Loops"	5
0.1.6	Exercise: "Write a function"	5
0.1.7	Exercise: "Print Function"	5
0.2	Data Types	5
0.2.1	Exercise: "List Comprehensions"	5
0.2.2	Exercise: "Find the Runner-Up Score!"	6
0.2.3	Exercise: "Nested Lists"	6
0.2.4	Exercise: "Finding the percentage"	6
0.2.5	Exercise: "Lists"	7
0.2.6	Exercise: "Tuples"	7
0.3	Strings	7
0.3.1	Exercise: "sWAP cASE"	7
0.3.2	Exercise: "String Split and Join"	8
0.3.3	Exercise: "What's Your Name?"	8
0.3.4	Exercise: "Mutations"	8
0.3.5	Exercise: "Find a string"	8
0.3.6	Exercise: "String Validators"	9
0.3.7	Exercise: "Text Alignment"	9
0.3.8	Exercise: "Text Wrap"	9
0.3.9	Exercise: "Designer Door Mat"	10
0.3.10	Exercise: "String Formatting"	10
0.3.11	Exercise: "Alphabet Rangoli"	10
0.3.12	Exercise: "Capitalize!"	11
0.3.13	Exercise: "Merge the Tools!"	11
0.3.14	Exercise: "The Minion Game"	11
0.4	Sets	11
0.4.1	Exercise: "Introduction to Sets"	11
0.4.2	Exercise: "No Idea!"	12
0.4.3	Exercise: "Symmetric Difference"	12
0.4.4	Exercise: "Set .add()"	12

0.4.5	Exercise: "Set .discard(), .remove() & .pop()"	12
0.4.6	Exercise: "Set .union() Operation"	13
0.4.7	Exercise: "Set .intersection() Operation"	13
0.4.8	Exercise: "Set .difference() Operation"	13
0.4.9	Exercise: "Set .symmetric_difference() Operation"	13
0.4.10	Exercise: "Set Mutations"	14
0.4.11	Exercise: "The Captain's Room"	14
0.4.12	Exercise: "Check Subset"	14
0.4.13	Exercise: "Check Strict Superset"	15
0.5	Collections	15
0.5.1	Exercise: "collections.Counter()"	15
0.5.2	Exercise: "DefaultDict Tutorial"	15
0.5.3	Exercise: "Collections.namedtuple()"	16
0.5.4	Exercise: "Collections.OrderedDict()"	16
0.5.5	Exercise: "Word Order"	17
0.5.6	Exercise: "Collections.deque()"	17
0.5.7	Exercise: "Company Logo"	18
0.5.8	Exercise: "Piling Up!"	18
0.6	Date and Time	18
0.6.1	Exercise: "Calendar Module"	18
0.6.2	Exercise: "Time Delta"	19
0.7	Exceptions	20
0.7.1	Exercise: "Exceptions"	20
0.8	Built-Ins	20
0.8.1	Exercise: "Zipped!"	20
0.8.2	Exercise: "Athlete Sort"	20
0.8.3	Exercise: "ginortS"	21
0.9	Python Functionals	21
0.9.1	Exercise: "Map and Lambda Function"	21
0.10	Regex and Parsing Challenges	22
0.10.1	Exercise: "Detect Floating Point Number"	22
0.10.2	Exercise: "Re.split()"	22
0.10.3	Exercise: "Group(), Groups() & Groupdict()"	22
0.10.4	Exercise: "Re.findall() & Re.finditer()"	22
0.10.5	Exercise: "Re.start() & Re.end()"	23
0.10.6	Exercise: "Regex Substitution"	23
0.10.7	Exercise: "Validating Roman Numerals"	23
0.10.8	Exercise: "Validating phone numbers"	24
0.10.9	Exercise: "Validating and Parsing Email Addresses"	24
0.10.10	Exercise: "Hex Color Code"	24
0.10.11	Exercise: "HTML Parser - Part 1"	25
0.10.12	Exercise: "HTML Parser - Part 2"	25
0.10.13	Exercise: "Detect HTML Tags, Attributes and Attribute Values"	26
0.10.14	Exercise: "Validating UID"	26
0.10.15	Exercise: "Validating Credit Card Numbers"	26
0.10.16	Exercise: "Validating Postal Codes"	27
0.10.17	Exercise: "Matrix Script"	27
0.11	XML	27

0.11.1	Exercise: "XML 1 - Find the Score"	27
0.11.2	Exercise: "XML2 - Find the Maximum Depth"	28
0.12	Closure and Decorations	28
0.12.1	Exercise: "Standardize Mobile Number Using Decorators"	28
0.12.2	Exercise: "Decorators 2 - Name Directory"	28
0.13	Numpy	28
0.13.1	Exercise: "Arrays"	28
0.13.2	Exercise: "Shape and Reshape"	29
0.13.3	Exercise: "Transpose and Flatten"	29
0.13.4	Exercise: "Concatenate"	29
0.13.5	Exercise: "Zeros and Ones"	29
0.13.6	Exercise: "Eye and Identity"	30
0.13.7	Exercise: "Array Mathematics"	30
0.13.8	Exercise: "Floor, Ceil and Rint"	31
0.13.9	Exercise: "Sum and Prod"	31
0.13.10	Exercise: "Min and Max"	31
0.13.11	Exercise: "Mean, Var, and Std"	31
0.13.12	Exercise: "Dot and Cross"	32
0.13.13	Exercise: "Inner and Outer"	32
0.13.14	Exercise: "Polynomials"	32
0.13.15	Exercise: "Linear Algebra"	33
1	Problem 2	33
1.1	Exercise: "Birthday Cake Candles"	33
1.2	Exercise: "Number Line Jumps"	33
1.3	Exercise: "Viral Advertising"	34
1.4	Exercise: "Recursive Digit Sum"	35
1.5	Exercise: "Insertion Sort - Part 1"	36
1.6	Exercise: "Insertion Sort - Part 2"	36

Problem 1

0.1 Introduction

Exercise: "Say 'Hello, World!' With Python"

```
1 if __name__ == '__main__':  
2     print("Hello, World!")
```

Exercise: "Python If-Else"

```
1 #!/bin/python3  
2  
3 import math  
4 import os  
5 import random  
6 import re  
7 import sys  
8  
9  
10  
11 if __name__ == '__main__':  
12     n = int(input().strip())  
13  
14     if n % 2 != 0:  
15         print('Weird')  
16     elif n % 2 == 0 and n >= 6 and n <= 20:  
17         print('Weird')  
18     else:  
19         print('Not Weird')
```

Exercise: "Arithmetic Operators"

```
1 if __name__ == '__main__':  
2     a = int(input())  
3     b = int(input())  
4  
5 c = a+b  
6 print(c)  
7 d = a-b  
8 print(d)  
9 e = a*b  
10 print(e)
```

Exercise: "Python: Division"

```
1 if __name__ == '__main__':  
2     a = int(input())  
3     b = int(input())  
4  
5 c = a//b  
6 d = a/b  
7
```

```
8 print(c)
9 print(d)
```

Exercise: "Loops"

```
1 if __name__ == '__main__':
2     n = int(input())
3
4     i = 0
5
6     while i < n:
7         print(i*i)
8         i += 1
```

Exercise: "Write a function"

```
1 def is_leap(year):
2     leap = False
3
4     if year % 4 == 0:
5         leap = True
6
7     if year % 100 == 0:
8         leap = False
9
10    if year % 400 == 0:
11        leap = True
12
13
14    return leap
```

Exercise: "Print Function"

```
1 if __name__ == '__main__':
2     n = int(input())
3
4     a = ''
5
6     for i in range(1, n+1):
7         a += str(i)
8
9     print(a)
```

0.2 Data Types

Exercise: "List Comprehensions"

```
1 if __name__ == '__main__':
2     x = int(input())
3     y = int(input())
4     z = int(input())
5     n = int(input())
```

```
6
7 def result(x, y, z, n):
8     coordinates = [[i, j, k] for i in range(x+1) for j in range(y+1) for
9                     k in range(z+1) if i+j+k != n]
10    print(coordinates)
11 result(x,y,z,n)
```

Exercise: "Find the Runner-Up Score!"

```
1 if __name__ == '__main__':
2     n = int(input())
3     arr = map(int, input().split())
4
5
6 arr = list(arr)
7 arr.sort(reverse=True)
8 arr = list(dict.fromkeys(arr))
9
10 print(arr[1])
```

Exercise: "Nested Lists"

```
1 if __name__ == '__main__':
2
3     list1 = []
4
5     for _ in range(int(input())):
6         list0 = []
7         name = input()
8         list0.append(name)
9         score = float(input())
10        list0.append(score)
11        list1.append(list0)
12        list0 = []
13
14    list1 = sorted(list1, key=lambda x: x[0])
15
16    min_value = min(list1, key=lambda x: x[1])[1]
17    second_lowest = min([x for x in list1 if x[1] != min_value], key=
18                        lambda x: x[1])[1]
19
20    for i in list1:
21        if i[1] == second_lowest:
22            print(i[0])
```

Exercise: "Finding the percentage"

```
1 if __name__ == '__main__':
2     n = int(input())
3     student_marks = {}
4     for _ in range(n):
5         name, *line = input().split()
6         scores = list(map(float, line))
```

```

7         student_marks[name] = scores
8     query_name = input()
9
10    for i in student_marks:
11        if i == query_name:
12            print("{:.2f}".format(sum(student_marks[i])/len(
                student_marks[i])))

```

Exercise: "Lists"

```

1  if __name__ == '__main__':
2      n = int(input())
3      list = []
4
5      for i in range(n):
6          command = input().split()
7          a = command[0]
8
9          if a == "insert":
10             pos = int(command[1])
11             val = int(command[2])
12             list.insert(pos, val)
13         elif a == "print":
14             print(list)
15         elif a == "remove":
16             val = int(command[1])
17             list.remove(val)
18         elif a == "append":
19             val = int(command[1])
20             list.append(val)
21         elif a == "sort":
22             list.sort()
23         elif a == "pop":
24             list.pop()
25         elif a == "reverse":
26             list.reverse()

```

Exercise: "Tuples"

```

1  if __name__ == '__main__':
2      n = int(input())
3      integer_list = map(int, input().split())
4      t = tuple(integer_list)
5      print(hash(t))

```

0.3 Strings

Exercise: "sWAP cASE"

```

1  def swap_case(s):
2      s1 = ""
3      for i in s:
4          if i.isupper():

```

```
5         s1 += i.lower()
6
7     elif i.islower():
8         s1 += i.upper()
9
10    else:
11        s1 += i
12
13    return s1
```

Exercise: "String Split and Join"

```
1 def split_and_join(line):
2     a = line.split()
3     a1 = "-".join(a)
4     return a1
5
6
7 if __name__ == '__main__':
8     line = input()
9     result = split_and_join(line)
10    print(result)
```

Exercise: "What's Your Name?"

```
1 def print_full_name(first, last):
2     print(f"Hello {first_name} {last_name}! You just delved into python.
3     ")
```

Exercise: "Mutations"

```
1 def mutate_string(string, position, character):
2     list_string = list(string)
3     list_string[position] = character
4     string = ''.join(list_string)
5
6     return string
```

Exercise: "Find a string"

```
1 def count_substring(string, sub_string):
2     count = 0
3     for i in range(len(string) - len(sub_string) + 1):
4         if string[i:i+len(sub_string)] == sub_string:
5             count += 1
6     return count
```


Exercise: "String Validators"

```
1 if __name__ == '__main__':
2     true_alphanumeric = False
3     true_alphabetical = False
4     true_digit = False
5     true_lowercase = False
6     true_uppercase = False
7
8     s = input()
9     for i in s:
10        if i.isalnum():
11            true_alphanumeric = True
12        if i.isalpha():
13            true_alphabetical = True
14        if i.isdigit():
15            true_digit = True
16        if i.islower():
17            true_lowercase = True
18        if i.isupper():
19            true_uppercase = True
20
21    print(true_alphanumeric)
22    print(true_alphabetical)
23    print(true_digit)
24    print(true_lowercase)
25    print(true_uppercase)
```

Exercise: "Text Alignment"

```
1 thickness = int(input())
2
3 for i in range(thickness):
4     print(('H' * i).rjust(thickness - 1) + 'H' + ('H' * i).ljust(
5         thickness - 1))
6
7 for i in range(thickness + 1):
8     print(('H' * thickness).center(thickness * 2) + ('H' * thickness).
9         center(thickness * 6))
10
11 for i in range((thickness + 1) // 2):
12     print(('H' * thickness * 5).center(thickness * 6))
13
14 for i in range(thickness + 1):
15     print(('H' * thickness).center(thickness * 2) + ('H' * thickness).
16         center(thickness * 6))
17
18 for i in range(thickness):
19     print(((('H' * (thickness - i - 1)).rjust(thickness) + 'H' + ('H' * (
20         thickness - i - 1)).ljust(thickness)).rjust(
21         thickness * 6))
```

Exercise: "Text Wrap"

```

1 def wrap(string, max_width):
2     list_wrap = textwrap.wrap(string, max_width)
3     strings = "\n".join(list_wrap)
4     return strings

```

Exercise: "Designer Door Mat"

```

1 n, m = map(int, input().split())
2 n1 = (n-1)/2
3
4 for i in range(int(n1)):
5     print(('.|.' * (2 * i + 1)).center(m, '-'))
6
7 print('WELCOME'.center(m, '-'))
8
9 for i in reversed(range(int(n1))):
10    print(('.|.' * (2 * i + 1)).center(m, '-'))

```

Exercise: "String Formatting"

```

1 def print_formatted(number):
2     width = len(bin(number)[2:])
3
4     for i in range(1, number+1):
5         decimal = str(i).rjust(width)
6         octal = oct(i)[2:].rjust(width)
7         hexa = hex(i)[2:].upper().rjust(width)
8         binary = bin(i)[2:].rjust(width)
9         print(decimal, octal, hexa, binary)
10
11
12 print(happiness)

```

Exercise: "Alphabet Rangoli"

```

1 alphabet = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l",
2             "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"]
3
4 def print_rangoli(size):
5     letters_tbu = alphabet[0:size][::-1]
6     pattern = []
7
8     for i in range(size):
9         row = "-".join(letters_tbu[:i] + letters_tbu[i::-1])
10        pattern.append(row.center(size * 4 - 3, "-"))
11
12    for i in range(size - 2, -1, -1):
13        pattern.append(pattern[i])
14
15    print("\n".join(pattern))

```

Exercise: "Capitalize!"

```
1 def solve(s):
2     list = re.split(r'(\s+)', s)
3     cap = ""
4     for i in list:
5         cap += i.capitalize()
6     return cap.strip()
```

Exercise: "Merge the Tools!"

```
1 def merge_the_tools(string, k):
2     parts = int(len(string)/k)
3
4     for i in range(0, parts):
5         what_to_print = ''
6         string_part = string[(i*k): k+(i*k)]
7         for i in string_part:
8             if i not in what_to_print:
9                 what_to_print += i
10        print(what_to_print)
```

Exercise: "The Minion Game"

```
1 def minion_game(string):
2     points_s = 0
3     points_k = 0
4     vowels = ['A', 'E', 'I', 'O', 'U']
5
6
7     for i in range(len(string)):
8         if string[i] in vowels:
9             points_k += len(string) - i
10        else:
11            points_s += len(string) - i
12
13    if points_s > points_k:
14        print("Stuart", points_s)
15    elif points_s < points_k:
16        print("Kevin", points_k)
17    else:
18        print("Draw")
```

0.4 Sets**Exercise: "Introduction to Sets"**

```
1 def average(array):
2     new_set = set(set(array))
3     avg = sum(new_set)/(len(new_set))
4     return "{:.3f}".format(avg)
```

Exercise: "No Idea!"

```
1 n, m = map(int, input().split())
2 X = list(map(int, input().split()))
3 A = set(map(int, input().split()))
4 B = set(map(int, input().split()))
5 happiness = 0
6
7 for i in range(len(X)):
8     if X[i] in A:
9         happiness += 1
10    elif X[i] in B:
11        happiness -= 1
12
13 print(happiness)
```

Exercise: "Symmetric Difference"

```
1 m = int(input())
2 set_m = set(map(int, input().split()))
3 n = int(input())
4 set_n = set(map(int, input().split()))
5
6 union = set_m.union(set_n)
7 distinct = union.difference(set_m.intersection(set_n))
8
9 distinct_list = list(distinct)
10 distinct = set(sorted(distinct_list))
11
12 for i in distinct:
13     print(i)
```

Exercise: "Set .add()"

```
1 n = int(input())
2 s = set()
3
4 for i in range(n):
5     country = input()
6     s.add(country)
7
8 print(len(s))
```

Exercise: "Set .discard(), .remove() & .pop()"

```
1 n = int(input())
2 s = set(map(int, input().split()))
3 n_command = int(input())
4 s_list = list(s)
5
6 for i in range(n_command):
7     command = input().split()
8     if command[0] == "pop" and len(s) > 0:
```

```
9         s_list.pop(0)
10     elif command[0] == "remove" or command[0] == "discard":
11         v = int(command[1])
12         if v in s_list:
13             s_list.remove(v)
14         else:
15             None
16
17 s = set(s_list)
18 print(sum(s))
```

Exercise: "Set .union() Operation"

```
1 n_english = int(input())
2 enrolled_english = set(map(int, input().split()))
3 n_french = int(input())
4 enrolled_french = set(map(int, input().split()))
5
6 print(len(enrolled_english.union(enrolled_french)))
```

Exercise: "Set .intersection() Operation"

```
1 n_english = int(input())
2 enrolled_english = set(map(int, input().split()))
3 n_french = int(input())
4 enrolled_french = set(map(int, input().split()))
5
6 print(len(enrolled_english.intersection(enrolled_french)))
```

Exercise: "Set .difference() Operation"

```
1 n_english = int(input())
2 enrolled_english = set(map(int, input().split()))
3 n_french = int(input())
4 enrolled_french = set(map(int, input().split()))
5
6 print(len(enrolled_english.difference(enrolled_french)))
```

Exercise: "Set .symmetric_difference() Operation"

```
1 n_english = int(input())
2 enrolled_english = set(map(int, input().split()))
3 n_french = int(input())
4 enrolled_french = set(map(int, input().split()))
5
6 print(len(enrolled_english.symmetric_difference(enrolled_french)))
```

Exercise: "Set Mutations"

```
1 n = int(input())
2 elements = set(map(int, input().split()))
3 N = int(input())
4
5 for i in range(N):
6     commands = input().split()
7     command = commands[0]
8     B = set(map(int, input().split()))
9     if command == "update":
10         elements.update(B)
11     elif command == "intersection_update":
12         elements.intersection_update(B)
13     elif command == "symmetric_difference_update":
14         elements.symmetric_difference_update(B)
15     elif command == "difference_update":
16         elements.difference_update(B)
17
18 print(sum(elements))
```

Exercise: "The Captain's Room"

```
1 size_group = int(input())
2 rooms = list(map(int, input().split()))
3 unique_rooms = set()
4 repeated_rooms = set()
5
6 for room in rooms:
7     if room in unique_rooms:
8         repeated_rooms.add(room)
9     else:
10        unique_rooms.add(room)
11
12 captain_room_number = sum(unique_rooms.difference(repeated_rooms))
13
14 print(captain_room_number)
```

Exercise: "Check Subset"

```
1 n_tc = int(input())
2
3 for i in range(n_tc):
4     n_a = int(input())
5     a = set(map(int, input().split()))
6     n_b = int(input())
7     b = set(map(int, input().split()))
8
9
10    if n_a == len(a.intersection(b)):
11        print(True)
12    else:
13        print(False)
```

Exercise: "Check Strict Superset"

```
1 a = set(map(int, input().split()))
2 n = int(input())
3 record = []
4 answer = 'True'
5
6 for i in range(n):
7     b = set(map(int, input().split()))
8     # print(len(a))
9     # print(len(a.intersection(b)))
10    if len(a) > len(a.intersection(b)) and len(b) == len(a.intersection(
11        b)):
12        record.append('True')
13    else:
14        record.append('False')
15
16 # print(record)
17
18
19 for x in record:
20     # print(x)
21     if x == 'False':
22         answer = 'False'
23         break
24
25 print(answer)
```

0.5 Collections**Exercise: "collections.Counter()"**

```
1 from collections import Counter
2
3 n_shoes = int(input())
4 sizes_available = list(map(int, input().split()))
5 n_customers = int(input())
6 revenue = 0
7
8 for i in range(n_customers):
9     shoe_desired = list(map(int, input().split()))
10    size = shoe_desired[0]
11    price = shoe_desired[1]
12
13    if size in sizes_available:
14        revenue += price
15        sizes_available.remove(size)
16    else:
17        revenue += 0
18
19 print(revenue)
```

Exercise: "DefaultDict Tutorial"

```
1 from collections import defaultdict
2
3 n, m = map(int, input().split())
4 A = defaultdict(list)
5 B = []
6
7 for i in range(1, n + 1):
8     a = input().strip()
9     A[a].append(i)
10
11 for i in range(m):
12     b = input().strip()
13     B.append(b)
14
15 for i in B:
16     if i in A:
17         positions_B = A[i]
18         print(*positions_B)
19     else:
20         print(-1)
```

Exercise: "Collections.namedtuple()"

```
1 from collections import namedtuple
2
3 n = int(input())
4 columns = list(map(str, input().split()))
5 student = namedtuple('student', columns)
6 students = []
7 sum_marks = 0
8
9 for i in range(n):
10     info = input().split()
11     student_instance = student(*info)
12     students.append(student_instance)
13
14 for i in students:
15     sum_marks += int(i.MARKS)
16
17 print(sum_marks/n)
```

Exercise: "Collections.OrderedDict()"

```
1 from collections import OrderedDict
2
3 n = int(input())
4 orders = OrderedDict()
5
6 for i in range(n):
7     info = input().split()
8     item = " ".join(info[:-1])
9     price = int(info[-1])
10     if item in orders:
11         orders[item] += price
```



```
12     else:
13         orders[item] = price
14
15 for i, p in orders.items():
16     print(i, p)
```

Exercise: "Word Order"

```
1 from collections import OrderedDict
2
3 n = int(input())
4 words = []
5
6 for i in range(n):
7     string = input()
8     words.append(string)
9
10 words_set = set(words)
11 word_ordered_dict = OrderedDict()
12
13 for i in words:
14     if i in word_ordered_dict:
15         word_ordered_dict[i] += 1
16     else:
17         word_ordered_dict[i] = 1
18
19
20 print(len(words_set))
21 print(*word_ordered_dict.values())
```

Exercise: "Collections.deque()"

```
1 from collections import deque
2
3 d=deque()
4 n=int(input())
5
6 for i in range(n):
7     input_list = list(map(str, input().split()))
8     command = input_list[0]
9     if command == 'append':
10         value = int(input_list[1])
11         d.append(value)
12     if command == 'appendleft':
13         value = int(input_list[1])
14         d.appendleft(value)
15     if command == 'pop':
16         d.pop()
17     if command == 'popleft':
18         d.popleft()
19
20 print(*d)
```

Exercise: "Company Logo"

```
1 import math
2 import os
3 import random
4 import re
5 import sys
6 from collections import Counter
7
8 def company_logo(string):
9     list_string = []
10    itera = 0
11    for i in string:
12        list_string.append(i)
13        counter = Counter(list_string)
14    for i in counter:
15        sorted_chars = sorted(counter.items(), key=lambda x: (-x[1], x
16                               [0]))
17        # found this way of sorting numbers and str online
18    for char, count in sorted_chars[:3]:
19        print(f"{char} {count}")
20
21 if __name__ == '__main__':
22     s = input()
23     company_logo(s)
```

Exercise: "Piling Up!"

```
1 from collections import deque
2
3 T = int(input())
4
5 for i in range(T):
6     n_block = int(input())
7     block = list(map(int, input().split()))
8     d = deque()
9     d.extend(block)
10    order = []
11    while d:
12        if d[0] >= d[-1]:
13            order.append(d[0])
14            d.popleft()
15        else:
16            order.append(d[-1])
17            d.pop()
18    if order == sorted(block, reverse=True):
19        print("Yes")
20    else:
21        print("No")
```

0.6 Date and Time

Exercise: "Calendar Module"

```
1 import calendar
2
3 month, day, year = map(int, input().split())
4 x = calendar.weekday(year, month, day)
5 if x == 0:
6     print("MONDAY")
7 elif x == 1:
8     print("TUESDAY")
9 elif x == 2:
10    print("WEDNESDAY")
11 elif x == 3:
12    print("THURSDAY")
13 elif x == 4:
14    print("FRIDAY")
15 elif x == 5:
16    print("SATURDAY")
17 elif x == 6:
18    print("SUNDAY")
19 else:
20    print("Invalid day of the week")
```

Exercise: "Time Delta"

```
1 import math
2 import os
3 import random
4 import re
5 import sys
6 from datetime import datetime
7
8 # Complete the time_delta function below.
9 def time_delta(t1, t2):
10     t1=datetime.strptime(t1, "%a %d %b %Y %H:%M:%S %z")
11     t2=datetime.strptime(t2, "%a %d %b %Y %H:%M:%S %z")
12     delta_t=t1-t2
13
14     difference=delta_t.total_seconds()
15     difference=abs(round(int(difference),0))
16     return str(difference)
17
18 if __name__ == '__main__':
19     fptr = open(os.environ['OUTPUT_PATH'], 'w')
20
21     t = int(input())
22
23     for t_itr in range(t):
24         t1 = input()
25
26         t2 = input()
27
28         delta = time_delta(t1, t2)
29
30         fptr.write(delta + '\n')
31
32     fptr.close()
```

0.7 Exceptions

Exercise: "Exceptions"

```
1 n = int(input())
2
3 for i in range(n):
4     try:
5         a, b = map(int, input().split())
6         print(int(a/b))
7     except ZeroDivisionError as e:
8         print("Error Code: integer division or modulo by zero")
9     except ValueError as e:
10        print("Error Code:", e)
```

0.8 Built-Ins

Exercise: "Zipped!"

```
1 N, X = map(int, input().split())
2 marks = []
3
4 for i in range(X):
5     marks.append(list(map(float, input().split())))
6
7 for i in zip(*marks):
8     num = sum(i)
9     den = len(i)
10    print(float(num/den))
```

Exercise: "Athlete Sort"

```
1 import math
2 import os
3 import random
4 import re
5 import sys
6
7
8 if __name__ == '__main__':
9     nm = input().split()
10
11     n = int(nm[0])
12
13     m = int(nm[1])
14
15     arr = []
16
17     for _ in range(n):
18         arr.append(list(map(int, input().rstrip().split())))
19
20     k = int(input())
21     sorted_data = sorted(arr, key=lambda x: x[k])
22
23     for i in sorted_data:
```

```
24 print(' '.join(map(str, i)))
```

Exercise: "ginortS"

```
1 lower_case = []
2 upper_case = []
3 numbers = []
4 odd = []
5 even = []
6
7 string = input()
8
9 for i in string:
10     if i.islower():
11         lower_case.append(i)
12     elif i.isupper():
13         upper_case.append(i)
14     else:
15         numbers.append(int(i))
16
17 for x in numbers:
18     if x%2 == 0:
19         even.append(x)
20     else:
21         odd.append(x)
22
23
24 sorted_lower_case = sorted(lower_case)
25 sorted_upper_case = sorted(upper_case)
26 sorted_odd = sorted(odd)
27 sorted_even = sorted(even)
28
29 final = sorted_lower_case + sorted_upper_case + sorted_odd + sorted_even
30 print(' '.join(map(str, final)))
```

0.9 Python Functionals

Exercise: "Map and Lambda Function"

```
1 def fibonacci(n):
2     fibonacci_numbers = []
3     for i in range(n):
4         if i == 0:
5             fibonacci_numbers.append(i)
6         elif i == 1:
7             fibonacci_numbers.append(i)
8         else:
9             fibonacci_numbers.append(fibonacci_numbers[i-2]+
10                                     fibonacci_numbers[i-1])
11     return fibonacci_numbers
12
13 cube = lambda x: x**3
```

0.10 Regex and Parsing Challenges

Exercise: "Detect Floating Point Number"

```
1 T = int(input())
2
3
4 for i in range(T):
5     try:
6         N = float(input())
7         if N == 0:
8             print(False)
9         else:
10            print(True)
11    except ValueError as e:
12        print(False)
13
14 float(+.5486468)
```

Exercise: "Re.split()"

```
1 regex_pattern = r"[,\.\]" # Do not delete 'r'.
```

Exercise: "Group(), Groups() & Groupdict()"

```
1 import re
2
3 S = list(input())
4
5 for i in range(len(S)):
6     if i < len(S)-1:
7         if S[i] == S[i+1] and S[i].isalnum():
8             print(S[i])
9             break
10    else:
11        print(-1)
12        break
```

Exercise: "Re.findall() & Re.finditer()"

```
1 import re
2
3 S = input()
4 S_list = list(S)
5
6 pattern = r'[aeiouAEIOU]{2,}'
7
8 rip = re.findall(pattern, S)
9
10 if S_list[len(S_list)-1].lower() not in ['a', 'e', 'i', 'o', 'u']:
11     for i in range(len(rip)):
12         print(rip[i])
13     if len(rip) == 0:
```

```

14         print(-1)
15
16 elif S_list[len(S_list)-1].lower() in ['a', 'e', 'i', 'o', 'u']:
17     for i in range(len(rip)-1):
18         print(rip[i])
19     if len(rip) == 0:
20         print(-1)

```

Exercise: "Re.start() & Re.end()"

```

1 import re
2
3 S = input()
4 k = input()
5 w = len(k)
6 count = 0
7
8
9 for i in range(len(S)):
10     try:
11         m = re.search(r"+" + k + "", S[i:i+w])
12         if m:
13             count += 1
14             print('(' + str(i+m.start()) + ', ' + str(i + m.end()-1) + '
15             ')')
16         except AttributeError:
17             pass
18
19 if count == 0:
20     print('(-1, -1)')

```

Exercise: "Regex Substitution"

```

1 import re
2
3
4 N = int(input())
5 html = ''
6
7
8 for i in range(N):
9     string = input() + " \n"
10    string = re.sub(r'(?<= )&&(?= )', 'and', string)
11    string = re.sub(r'(?<= )\\|\\|(?= )', 'or', string)
12    html += string
13
14 print(html)

```

Exercise: "Validating Roman Numerals"

```

1 regex_pattern = r"^M{0,3}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I
  {0,3})$" # Do not delete 'r'.

```

Exercise: "Validating phone numbers"

```
1 import re
2
3 N = int(input())
4 pattern = r'^[789]\d{9}$'
5 numbers = []
6
7 for i in range(N):
8     number = input()
9     if re.match(pattern, number):
10         print('YES')
11     else:
12         print('NO')
```

Exercise: "Validating and Parsing Email Addresses"

```
1 import re
2 import email.utils
3
4 N = int(input())
5 pattern = r'^[a-zA-Z][a-zA-Z0-9_.-]+@[a-zA-Z]+\.[a-zA-Z]{1,3}$'
6
7 for i in range(N):
8
9     nothing, address = email.utils.parseaddr(input())
10
11     if re.match(pattern, address):
12         print(nothing + " " + "<" + address + ">")
```

Exercise: "Hex Color Code"

```
1 import re
2
3 N = int(input())
4 string = ''
5
6 for i in range(N):
7     string_part = input()
8     string += string_part
9
10 new_string = string.replace('#BED', '')
11 new_string = new_string.replace('#Cab', '')
12 new_string = new_string.replace('#f0f', '')
13
14 pattern = r'#([0-9A-Fa-f]{6}|[0-9A-Fa-f]{3})\b'
15
16 matches = re.findall(pattern, new_string)
17
18 for i in matches:
19     print('#' + i)
```


Exercise: "HTML Parser - Part 1"

```

1 from html.parser import HTMLParser
2
3 N = int(input())
4 html = ''
5
6 class MyHTMLParser(HTMLParser):
7     def handle_starttag(self, tag, attrs):
8         print("Start :", tag)
9         for i in attrs:
10             atts_list = list(map(str, i))
11             print("-> " + atts_list[0] + " > " + atts_list[1])
12     def handle_endtag(self, tag):
13         print("End   :", tag)
14     def handle_startendtag(self, tag, attrs):
15         print("Empty :", tag)
16         for i in attrs:
17             atts_list = list(map(str, i))
18             print("-> " + atts_list[0] + " > " + atts_list[1])
19
20
21 for i in range(N):
22     string = input()
23     html += string
24
25 parser = MyHTMLParser()
26 parser.feed(html)

```

Exercise: "HTML Parser - Part 2"

```

1 from html.parser import HTMLParser
2 import re
3
4 class MyHTMLParser(HTMLParser):
5     def handle_data(self, data):
6         if data != '\n':
7             print(">>> Data")
8             print(data)
9
10    def handle_comment(self, data):
11        if bool(re.search(r"\n", data)) == True:
12            print(">>> Multi-line Comment")
13            print(data)
14        else:
15            print(">>> Single-line Comment")
16            print(data)
17
18    html = ""
19    for i in range(int(input())):
20        html += input().rstrip()
21        html += '\n'
22
23    parser = MyHTMLParser()
24    parser.feed(html)
25    parser.close()

```

Exercise: "Detect HTML Tags, Attributes and Attribute Values"

```

1 from html.parser import HTMLParser
2
3 class MyHTMLParser(HTMLParser):
4     def handle_starttag(self, tag, attrs):
5         print(tag)
6         for i in attrs:
7             atts_list = list(map(str, i))
8             print("> " + atts_list[0] + " > " + atts_list[1])
9
10    def handle_startendtag(self, tag, attrs):
11        print(tag)
12        for i in attrs:
13            atts_list = list(map(str, i))
14            print("> " + atts_list[0] + " > " + atts_list[1])
15
16    html = ""
17    N = int(input())
18    for i in range(N):
19        html += input().rstrip()
20        html += '\n'
21
22    parser = MyHTMLParser()
23    parser.feed(html)
24    parser.close()

```

Exercise: "Validating UID"

```

1 import re
2
3 N = int(input())
4
5 pattern = r'^(?=(?:[A-Z]*[A-Z]){2})(?=(?:\D*\d){3})(?!.*(.) .* \1)[A-Za-z0-9]{10}$'
6
7 # found this pattern on stackoverflow but I truly understood it
8
9
10 for i in range(N):
11     code = input()
12     if re.match(pattern, code):
13         print('Valid')
14     else:
15         print('Invalid')

```

Exercise: "Validating Credit Card Numbers"

```

1 import re
2
3 N = int(input())
4 pattern = r'^(?!.*(\d)(?:-\d\1){3})[4-6][0-9]{3}(-?[0-9]{4}){3}$'
5
6 for i in range(N):
7     card = input()

```

```

8     if re.match(pattern, card):
9         print('Valid')
10    else:
11        print('Invalid')

```

Exercise: "Validating Postal Codes"

```

1 regex_integer_in_range = r"^[1-9][0-9]{5}$" # Do not delete 'r'.
2 regex_alternating_repetitive_digit_pair = r"(\d)(?=\d\1)"

```

Exercise: "Matrix Script"

```

1 import math
2 import os
3 import random
4 import re
5 import sys
6
7
8 first_multiple_input = input().rstrip().split()
9
10 n = int(first_multiple_input[0])
11
12 m = int(first_multiple_input[1])
13
14 matrix = []
15
16 for _ in range(n):
17     matrix_item = input()
18     matrix.append(matrix_item)
19
20
21 string = ''
22
23 for i in range(m):
24     try:
25         for x in range(n):
26             string += matrix[x][i]
27     except IndexError:
28         pass
29
30 final = re.sub(r'(?<=[a-zA-Z0-9])\W+(?=[a-zA-Z0-9])', ' ', string)
31
32 print(final)

```

0.11 XML

Exercise: "XML 1 - Find the Score"

```

1 def get_attr_number(node):
2     count=0
3     for i in node.iter():
4         count += len(i.attrib)

```

```

5     return counted_people
6     return inner

```

Exercise: "XML2 - Find the Maximum Depth"

```

1 maxdepth = 0
2 levels = []
3
4 def depth(elem, level):
5     global maxdepth
6     if level == -1:
7         level += 1
8         levels.append(level)
9         maxdepth = max(levels)
10    for i in elem:
11        depth(i, level + 1)

```

0.12 Closure and Decorations

Exercise: "Standardize Mobile Number Using Decorators"

```

1 def wrapper(f):
2     def fun(l):
3         formatted_nb = []
4         formatted_nb_final = []
5         for number in sorted(l):
6             formatted_nb.append(number[-10:])
7         for number in sorted(formatted_nb):
8             formatted_nb_final.append('+91 ' + number[-10:-5] + ' ' +
9                                     number[-5:])
10        for i in formatted_nb_final:
11            print(i)
12    return fun

```

Exercise: "Decorators 2 - Name Directory"

```

1 def person_lister(f):
2     def inner(people):
3         list_people = []
4         for i in people:
5             list_people.append(i)
6         people = sorted(list_people, key = lambda x: int(x[2]))
7         formatted_people = []
8         for person in people:
9             formatted_people.append(f(person))
10
11    return formatted_people
12    return inner

```

0.13 Numpy

Exercise: "Arrays"

```
1 def arrays(arr):
2     b = arr[::-1]
3     a = numpy.array(b, float)
4     return a
```

Exercise: "Shape and Reshape"

```
1 import numpy as np
2
3 a = np.array(list(map(int, input().split())))
4 print(np.reshape(a, (3,3)))
```

Exercise: "Transpose and Flatten"

```
1 import numpy
2
3 n, m = map(int, input().split())
4 arr1 = []
5
6 for i in range(n):
7     arr = numpy.array(list(map(int, input().split())))
8     arr1.append(arr)
9
10 arr2 = numpy.array(arr1)
11
12 print(arr2.transpose())
```

Exercise: "Concatenate"

```
1 import numpy
2
3 N, M, P = map(int, input().split())
4 N_array = []
5 M_array = []
6
7 for i in range(N):
8     arr = numpy.array(list(map(int, input().split())))
9     N_array.append(arr)
10
11 for i in range(M):
12     arr = numpy.array(list(map(int, input().split())))
13     M_array.append(arr)
14
15 print(numpy.concatenate((N_array, M_array)))
```

Exercise: "Zeros and Ones"

```
1 import numpy as np
2
3 input_values = input().split()
4
```

```
5 if len(input_values) == 4:
6     X, Y, Z, B = map(int, input_values)
7     print(np.zeros((X, Y, Z, B), dtype = int))
8     print(np.ones((X, Y, Z, B), dtype = int))
9
10 elif len(input_values) == 3:
11     X, Y, Z = map(int, input_values)
12     print(np.zeros((X, Y, Z), dtype = int))
13     print(np.ones((X, Y, Z), dtype = int))
14
15 elif len(input_values) == 2:
16     X, Y = map(int, input_values)
17     print(np.zeros((X, Y), dtype = int))
18     print(np.ones((X, Y), dtype = int))
```

Exercise: "Eye and Identity"

```
1 import numpy
2
3
4 N, M = map(int, input().split())
5 arr = numpy.eye(N, M, k = 0)
6 numpy.set_printoptions(legacy = '1.13')
7
8 print(arr)
```

Exercise: "Array Mathematics"

```
1 import numpy
2
3
4 N, M = map(int, input().split())
5 A = []
6 B = []
7
8 for i in range(N):
9     arr = numpy.array(input().split(), int)
10    A.append(arr)
11
12 for i in range(N):
13     arr = numpy.array(input().split(), int)
14    B.append(arr)
15
16 A_arr = numpy.array(A, int).reshape(N, M)
17 B_arr = numpy.array(B, int).reshape(N, M)
18
19
20 print(numpy.add(A, B))
21 print(numpy.subtract(A, B))
22 print(numpy.multiply(A, B))
23 print(numpy.floor_divide(A, B))
24 print(numpy.mod(A, B))
25 print(numpy.power(A, B))
```

Exercise: "Floor, Ceil and Rint"

```
1 import numpy as np
2
3 input_arr = np.array(list(map(float, input().split())))
4
5 np.set_printoptions(legacy='1.13')
6
7 print(np.floor(input_arr))
8 print(np.ceil(input_arr))
9 print(np rint(input_arr))
```

Exercise: "Sum and Prod"

```
1 import numpy
2
3 N, M = map(int, input().split())
4 A = []
5
6 for i in range(N):
7     arr = numpy.array(list(map(int, input().split())))
8     A.append(arr)
9
10 added = numpy.sum(A, axis = 0)
11 print(numpy.prod(added))
```

Exercise: "Min and Max"

```
1 import numpy
2
3
4 N, M = map(int, input().split())
5 A = []
6
7 for i in range(N):
8     arr = numpy.array(list(map(int, input().split())))
9     A.append(arr)
10
11 A_min = numpy.min(A, axis = 1)
12 print(numpy.max(A_min))
```

Exercise: "Mean, Var, and Std"

```
1 import numpy
2
3
4 N, M = map(int, input().split())
5 A = []
6
7 for i in range(N):
8     arr = numpy.array(list(map(int, input().split())))
9     A.append(arr)
10
```

```
11 A_mean = numpy.mean(A, axis = 1)
12 print(A_mean)
13 A_var = numpy.var(A, axis = 0)
14 print(A_var)
15 A_std = round(numpy.std(A), 11)
16 print(A_std)
```

Exercise: "Dot and Cross"

```
1 import numpy
2
3
4 N = int(input())
5 A = []
6 B = []
7
8 for i in range(N):
9     arr = numpy.array(list(map(int, input().split())))
10    A.append(arr)
11
12
13 for i in range(N):
14     arr = numpy.array(list(map(int, input().split())))
15    B.append(arr)
16
17 A_cross = numpy.dot(A, B)
18 print(A_cross)
```

Exercise: "Inner and Outer"

```
1 import numpy
2
3 A = numpy.array(list(map(int, input().split())))
4 B = numpy.array(list(map(int, input().split())))
5
6 inner = numpy.inner(A, B)
7 print(inner)
8
9 outer = numpy.outer(A, B)
10 print(outer)
```

Exercise: "Polynomials"

```
1 import numpy
2
3 P = numpy.array(list(map(float, input().split())))
4 x = int(input())
5
6 print(numpy.polyval(P, x))
```


Exercise: "Linear Algebra"

```
1 import numpy
2
3
4 N = int(input())
5 A = []
6
7 for i in range(N):
8     arr = numpy.array(list(map(float, input().split())))
9     A.append(arr)
10
11 print(round(numpy.linalg.det(A), 4))
```

1 Problem 2

1.1 Exercise: "Birthday Cake Candles"

```
1 import math
2 import os
3 import random
4 import re
5 import sys
6
7 def birthdayCakeCandles(candles):
8     max_height = max(candles)
9     count = 0
10    for i in candles:
11        if i == max_height:
12            count += 1
13    return count
14
15 if __name__ == '__main__':
16     fptr = open(os.environ['OUTPUT_PATH'], 'w')
17
18     candles_count = int(input().strip())
19
20     candles = list(map(int, input().rstrip().split()))
21
22     result = birthdayCakeCandles(candles)
23
24     fptr.write(str(result) + '\n')
25
26     fptr.close()
```

1.2 Exercise: "Number Line Jumps"

```
1 import math
2 import os
3 import random
4 import re
5 import sys
6
```

```
7 def kangaroo(x1, v1, x2, v2):
8     answer = 'NO'
9     if x1 == x2:
10         answer = 'YES'
11
12
13     elif x1 > x2 and v2 > v1:
14         while x1 + v1 >= x2 + v2:
15             x1 += v1
16             x2 += v2
17             if x1 == x2:
18                 answer = 'YES'
19
20
21     elif x2 > x1 and v1 > v2:
22         while x2 + v2 >= x1 + v1:
23             x1 += v1
24             x2 += v2
25             if x1 == x2:
26                 answer = 'YES'
27
28     return answer
29
30
31 if __name__ == '__main__':
32     fptr = open(os.environ['OUTPUT_PATH'], 'w')
33
34     first_multiple_input = input().rstrip().split()
35
36     x1 = int(first_multiple_input[0])
37
38     v1 = int(first_multiple_input[1])
39
40     x2 = int(first_multiple_input[2])
41
42     v2 = int(first_multiple_input[3])
43
44     result = kangaroo(x1, v1, x2, v2)
45
46     fptr.write(result + '\n')
47
48     fptr.close()
```

1.3 Exercise: "Viral Advertising"

```
1 import math
2 import os
3 import random
4 import re
5 import sys
6
7 def viralAdvertising(n):
8     cumulative = 0
9     shared = 5
10
11     for i in range(1, n+1):
```

```

12         liked = math.floor(shared / 2)
13         cumulative += liked
14         shared = liked * 3
15
16     return cumulative
17
18 if __name__ == '__main__':
19     fptr = open(os.environ['OUTPUT_PATH'], 'w')
20
21     n = int(input().strip())
22
23     result = viralAdvertising(n)
24
25     fptr.write(str(result) + '\n')
26
27     fptr.close()

```

1.4 Exercise: "Recursive Digit Sum"

```

1 import math
2 import os
3 import random
4 import re
5 import sys
6
7 def superDigit(n, k):
8
9     n_list = list(n)
10
11     while len(n_list) != 1:
12         n_new = 0
13         for digit in n_list:
14             n_new += int(digit)
15         n_list = list(str(n_new))
16
17     n_str = ''.join(n_list)
18     n_final = int(n_str) * k
19     n_list_final = list(str(n_final))
20
21     while len(n_list_final) != 1:
22         n_new = 0
23         for digit in n_list_final:
24             n_new += int(digit)
25         n_list_final = list(str(n_new))
26
27     return int(n_list_final[0])
28
29 if __name__ == '__main__':
30     fptr = open(os.environ['OUTPUT_PATH'], 'w')
31
32     first_multiple_input = input().rstrip().split()
33
34     n = first_multiple_input[0]
35
36     k = int(first_multiple_input[1])
37

```

```
38     result = superDigit(n, k)
39
40     fptr.write(str(result) + '\n')
41
42     fptr.close()
```

1.5 Exercise: "Insertion Sort - Part 1"

```
1  import math
2  import os
3  import random
4  import re
5  import sys
6
7  def insertionSort1(n, arr):
8      arr_list = list(arr)
9      o_num = arr_list[int(len(arr_list)-1)]
10     for i in range(len(arr_list)-2, -1, -1):
11         if o_num < arr_list[i]:
12             arr_list[i + 1] = arr_list[i]
13             string_list = list(map(str, arr_list))
14             result = " ".join(string_list)
15             print(result)
16
17         else:
18             arr_list[i + 1] = o_num
19             string_list = list(map(str, arr_list))
20             result = " ".join(string_list)
21             print(result)
22             break
23
24     if o_num < arr_list[0]:
25         arr_list[0] = o_num
26         string_list = list(map(str, arr_list))
27         result = " ".join(string_list)
28         print(result)
29
30
31
32
33 if __name__ == '__main__':
34     n = int(input().strip())
35
36     arr = list(map(int, input().rstrip().split()))
37
38     insertionSort1(n, arr)
```

1.6 Exercise: "Insertion Sort - Part 2"

```
1  import math
2  import os
3  import random
4  import re
5  import sys
6
```

```
7 def insertionSort2(n, arr):
8     arr_list = list(arr)
9     for i in range(1, n):
10         key = arr_list[i]
11         j = i - 1
12         while j >= 0 and key < arr_list[j]:
13             arr_list[j + 1] = arr_list[j]
14             j -= 1
15         arr_list[j + 1] = key
16         print(" ".join(map(str, arr_list)))
17
18 if __name__ == '__main__':
19     n = int(input().strip())
20
21     arr = list(map(int, input().rstrip().split()))
22
23     insertionSort2(n, arr)
```