PYTHON CODES

Week 1

PPA 1

Print the first 5 positive integers in ascending order with one number in each line

```
for i in range (5):
  print(i+1)
```

PPA 2

Print the following pattern.

```
*
**
***
****
*****
```

There are no spaces between consecutive stars. There are no spaces at the end of each line.

```
for i in range (1,6):
  for j in range(1,i+1):
print("*", end = '')
print()
```

PPA 3

Accept an integer as input and print its square as output.

```
n = int(input())
print(n**2)
```

PPA 4

Accept two integers as input and print their sum as output.

```
a = int(input())
```

```
b = int(input())
print(a+b)
```

PPA 5

Accept two words as input and print the two words after adding a space between them.

```
print(input(), input())
```

PPA 6

Accept the registration number of a vehicle as input and print its state-code as output.

```
s = input()
print(s[0:2])
```

PPA 7

Accept a five-digit number as input and print the sum of its digits as output.

```
num = input()

d1 = int(num[0])
d2 = int(num[1])
d3 = int(num[2])
d4 = int(num[3])
d5 = int(num[4])

dsum = d1 + d2 + d3 + d4 + d5
print(dsum)
```

GrPA 1

Accept five words as input and print the sentence formed by these words after adding a space between consecutive words and a full stop at the end.

```
word1 = input()
word2 = input()
word3 = input()
word4 = input()
word5 = input()
space = ' '
stop = '.'
sentence = word1 + space + word2 + space + word3 + space + word4 + space +
word5 + stop
print(sentence)
```

GrPA 2

Accept the date in DD-MM-YYYY format as input and print the year as output.

```
date = input()
year = date[-4: ]
print(year)
```

GrPA 3

Accept a sequence of five single digit numbers separated by commas as input. Print the product of all five numbers.

```
num = input()

d1 = int(num[0])
d2 = int(num[2])
d3 = int(num[4])
d4 = int(num[6])
d5 = int(num[8])

dprod = d1 * d2 * d3 * d4 * d5
print(dprod)
```

GrPA 4

Assume that several IITs start offering online degrees across multiple branches. The email-id of a student is defined as follows:
branch_degree_year_roll@student.onlinedegree.institute.ac.in

For example, if the email-id is CS_BT_21_7412@student.onlinedegree.iitm.ac.in, then this student is from the computer science branch, pursuing a BTech degree from IITM, starting from the year 2021, with 7412 as the roll number. branch, degree and year are codes of length two, while roll and institute are codes of length four. Accept a student's email-id as input and print the following details, one item on each line: (1) Branch (2) Degree (3) Year (4) Roll number (5) Institute

```
email = input()
branch = email[:2]
degree = email[3: 5]
year = email[6:8]
roll = email[9:13]
institute = email[-10:-6]

print(branch)
print(degree)
print(year)
print(roll)
print(institute)
```

GrPA 5

Accept two positive integers x and y as input. Print the number of digits in x^y.

```
x = int(input())
y = int(input())
res = x ** y
res_str = str(res)
print(len(res_str))
```

GrPA 6

Accept two positive integers M and N as input. There are two cases to consider: (1) If $M < N$, then print M as output. (2) If $M >= N$, subtract N from M. Call the difference M_1. If $M_1 >= N$, then subtract N from M_1 and call the difference M_2. Keep doing this operation until you reach a value k, such that, $M_k < N$. You have to print the value of M_k as output.

```
M = int(input())
N = int(input())
print(M % N)
```

Week 2

PPA 1

Accept a non-zero integer as input. Print `positive` if it is greater than zero and `negative` if it is less than zero.

```
n = int(input() )
if n>0:
    print('positive')
else:
    print('negative')
```

PPA 2

Consider the piece-wise function given below.

$$f(x) = \begin{cases} x + 2 & 0 < x < 10 \\ x^2 + 2 & 10 \leq x \\ 0 & otherwise \end{cases}$$

Accept the value of *x* as input and print the value of *f*(*x*) as output. Note that both the input and output are real numbers. Your code should reflect this aspect. That is, both *x* and *f*(*x*) should be float values.

```
x = float(input())
if 0<x<10:
  print(x+2)
elif(10<=x):
  print(x**2+2)
else:
print(0)
```

PPA 3

Accept an integer as input and print the time of the day. Use the following table for reference.

Input	Output
$T < 0$	INVALID
$0 \leq T \leq 5$	NIGHT
$6 \leq T \leq 11$	MORNING
$12 \leq T \leq 17$	AFTERNOON
$18 \leq T \leq 23$	EVENING
$T \geq 24$	INVALID

The input will be a single line containing an integer. The output should be one of these strings: NIGHT, MORNING, AFTERNOON, EVENING, INVALID.

```
T = int(input())
if T<0:
    print('INVALID')
elif 0<=T<=5:
    print('NIGHT')
elif 6<=T<=11:
    print('MORNING')
elif 12<=T<=17:
    print('AFTERNOON')
elif 18<=T<=23:
    print('EVENING')
else:
    print('INVALID')
```

PPA 4

Accept a point in 2D space as input and find the region in space that this point belongs to. A point could belong to one of the four quadrants, or it could be on one of the two axes, or it could be the origin. The input is given in 2 lines: the first line is the x-coordinate of the point while the second line is its y-coordinate. The possible outputs are `first`, `second`, `third`, `fourth`, `x-axis`, `y-axis`, and `origin`. Any other output will not be accepted. Note that all outputs should be in lower case.

```
x = float(input())
y = float(input())
if x>0:
  if y>0:
    print("first")
elif y<0:
      print("fourth")
elif(y==0):
      print("x-axis")
if x<0:
  if y>0:
    print("second")
elif(y<0):
    print("third")
elif(y==0):
    print("x-axis")
if x==0:
  if y==0:
    print("origin")
  else:
    print("y-axis")
```

PPA 5

Write a program to realize the equation of a line given 2 points (x_1,y_1) and (x_2,y_2) in 2D space. The input is in 5 lines where, the first, second, third, and fourth lines represent x_1, y_1, x_2, and y_2 respectively. The fifth line corresponds to x_3.
Determine y_3 using the equation of a straight line as given below:

$$\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1}$$

The output should be "Vertical Line" if the line is vertical. In other cases, the output should be 2 lined, where the first line is the value of y_3 and the second line indicates whether the slope of the line is positive, negative or zero. Print "Positive Slope", "Negative Slope" or "Horizontal Line" accordingly.
Note that all inputs are to be processed as real numbers.

```
x1 = float(input())
y1 = float(input())
x2 = float(input())
y2 = float(input())
x3 = float(input())
```

```
if x1==x2:
print("Vertical Line")
else:
  slope = (y2-y1)/(x2-x1)
  y3 = y1 + slope*(x3-x1)
  print(y3)
  if slope>0:
print("Positive Slope")
elif slope<0:
print("Negative Slope")
  else:
print("Horizontal Line")
```

PPA 6

Accept a string as input. If the input string is of odd length, then continue with it. If the input string is of even length, make the string of odd length as below:

- If the last character is a period (`.`), then remove it
- If the last character is not a period, then add a period (`.`) to the end of the string

Call this string of odd length `word`. Select a substring made up of three consecutive characters from `word` such that there are an equal number of characters to the left and right of this substring. Print this substring as output. You can assume that all input strings will be in lower case and will have a length of at least four.

```
s = input()
n = len(s)
if n%2 == 0:
    if s[n-1]=='.':
        s=s[:-1]
    else:
        s=s+'.'
n = int((len(s)-1)/2)
print(s[n-1: n+2])
```

PPA 7

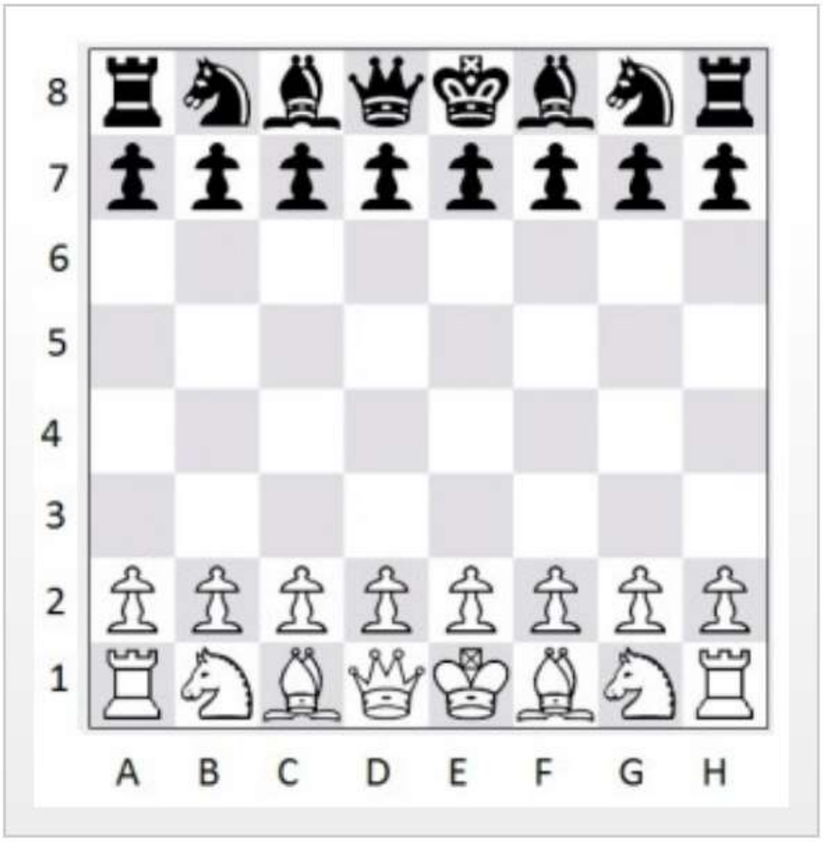
A sequence of five words is called **magical** if the i^th word is a substring of the (i+1)^th word for every i in range 1≤ i < 5. Accept a sequence of five words as input, one word on each line. Print ***magical*** if the sequence is magical and ***non-magical*** otherwise.

Note that str_1 is a substring of str_2 if and only if str_1 is present as a sequence of consecutive characters in str_2.

```python
a = input()
b = input()
c = input()
d = input()
e = input()
if(a in b and b in c and c in d and d in e):
    print("magical")
else:
    print("non-magical")
```

PPA 8

Consider the following image of a chessboard:



Accept two positions as input: start and end. Print YES if a bishop at start can move to end in exactly one move.

Print NO otherwise. Note that a bishop can only move along diagonals.

```
start = input()
end = input()
s = 'ABCDEFGH'
if(abs(s.index(start[0])-s.index(end[0])))==abs(int(start[1]) - int(end[1])):
    print('YES')
else:
    print('NO')
```

PPA 9

You have n gold coins with you. You wish to divide this among three of your friends under the following conditions:

(1) All three of them should get a non-zero share.

(2) No two of them should get the same number of coins.

(3) You should not have any coins with you at the end of this sharing process.

The input has four lines. The first line contains the number of coins with you. The next three lines will have the share given to your three friends. All inputs shall be non-negative integers. If the division satisfies these conditions, then print the string `FAIR`. If not, print `UNFAIR`.

```
n=int(input())
n1=int(input())
n2=int(input())
n3=int(input())
if n1>0 and n2>0 and n3>0 and n1+n2+n3==n and n1!=n2 and n2!=n3 and n3!=n1 :
    print('FAIR')
else:
    print('UNFAIR')
```

PPA 10

Accept a real number X as input and print the greatest integer less than or equal to X on the first line, followed by the smallest integer greater than or equal to X on the second line.

```
x = float(input())
n = int(x)
if x==n:
  print(n)
  print(n)
```

```
elif(x>0):
  print(n)
  print(n+1)
else:
  print(n-1)
  print(n)
```

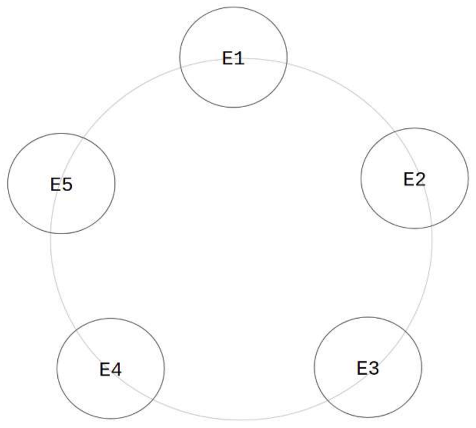
GrPA 1

Accept three positive integers as input and check if they form the sides of a right triangle. Print YES if they form one, and NO is they do not. The input will have three lines, with one integer on each line. The output should be a single line containing one of these two strings: YES or NO.

```
x = int(input())
y = int(input())
z = int(input())

# There are three possible conditions
# We are combining them using the "or" operator
if ((x ** 2 + y ** 2 == z ** 2) or
    (y ** 2 + z ** 2 == x ** 2) or
    (z ** 2 + x ** 2 == y ** 2)):
  print('YES')
else:
  print('NO')
```

GrPA 2

EvenOdd is a tech startup. Each employee at the startup is given an employee id which is a unique positive integer. On one warm Sunday evening, five employees of the company come together for a meeting and sit at a circular table:



The employees follow a strange convention. They will continue the meeting only if the following condition is satisfied.

The sum of the employee-ids of every pair of adjacent employees at the table must be an even number.

They are so lazy that they won't move around to satisfy the above condition, If the current seating plan doesn't satisfy the condition, the meeting will be cancelled. You are given the employee-id of all five employees. Your task is to decide if the meeting happened or not.

The input will be five lined, each containing an integer. The ith line will have the employee-id of Ei. The output will be a single line containing one of these two strings: YES or NO.

```
e1 = int(input())
e2 = int(input())
e3 = int(input())
e4 = int(input())
e5 = int(input())
# Check if the sum is odd for each pair of adjacent employees
if (e1 + e2) % 2 != 0:
    print('NO')
elif (e2 + e3) % 2 != 0:
    print('NO')
elif (e3 + e4) % 2 != 0:
    print('NO')
elif (e4 + e5) % 2 != 0:
    print('NO')
elif (e5 + e1) % 2 != 0:
    print('NO')
# If the sum is even for every pair of adjacent employees,
# then the else block gets executed
else:
    print('YES')
```

GrPA 3

Accept a string as input and print the vowels present in the string in alphabetical order. If the string doesn't contain any vowels, then print the string ***none*** as output. Each vowel that appears in the input string – irrespective of its case should appear just once in lower case in the output.

```python
input_string = input().lower()
vowels = ""
if "a" in input_string:
    vowels += "a"
if "e" in input_string:
    vowels += "e"
if "i" in input_string:
    vowels += "i"
if "o" in input_string:
    vowels += "o"
if "u" in input_string:
    vowels += "u"
# check if vowels is non-empty
if vowels != "":
    print(vowels)
else:
    print('none')
```

GrPA 4

You are given the date of birth of two persons, not necessarily from the same family. Your task is to find the younger of the two. If both of them share the same date of birth, then the younger of the two is assumed to be that person whose name comes first in alphabetical order.

The input will have four lines. The first two lines correspond to the first person, while the last two lines correspond to the second person. For each person, the first line corresponds to the name and the second line corresponds to the date of birth in "DD-MM-YYYY" format. Your output should be the name of the younger of the two.

```
n1 = input()
d1 = input()
n2 = input()
d2 = input()
if d1==d2 :
    if n1<n2 :
        print(n1)
else :
        print(n2)
elif d1[-4:] != d2[-4:] :
    if int(d1[-4:]) < int(d2[-4:]):
        print(n2)
else :
        print(n1)
elif d1[3:5] != d2[3:5]:
    if int(d1[3:5]) < int(d2[3:5]):
        print(n2)
else :
        print(n1)
else :
    if int(d1[0:2]) < int(d2[0:2]):
        print(n2)
else :
        print(n1)
```

GrPA 5

Accept a string as input. Your task is to determine if the input string is a valid password or not. For a string to be a valid password, it must satisfy all the conditions given below:

(1) It should have at least 8 and at most 32 characters

(2) It should start with an uppercase or lowercase letter

(3) It should not have any of these characters: / \ = ' "

(4) It should not have spaces

It could have any character that is not mentioned in the list of characters to be avoided (points 3 and 4). Output True if the string forms a valid password and False otherwise.

```
p = input()
if 8<= len(p) <= 32 and p[0].isalpha() and '/' not in p and '\\' not in p and
'=' not in p and '\'' not in p and '\"' not in p and ' ' not in p :
    print('True')
else:
    print('False')
```

Week 3

PPA 1

Accept a positive integer n as input and print the first n positive integers, one number on each line.

```
n = int(input())
for i in range(1,n+1):
    print(i)
```

PPA 2

Accept a positive integer n as input and print all the factors of n, one number on each line.

```
n = int(input())
for i in range(1,n+1):
    if n%i==0:
        print(i)
```

PPA 3

Accept two positive integers a and b as input. Print the sum of all integers in the range $[1000, 2000]$, endpoints inclusive, that are divisible by both a and b. If you find no number satisfying this condition in the given range, then print 0.

```
a = int(input())
b = int(input())
n = 0
for i in range(1000,2001):
    if i%a==0 and i%b==0:
        n+=i
print(n)
```

PPA 4

Accept a positive integer n as input, where n is greater than 1.

Print PRIME if n is a prime number and $\mathrm{NOTPRIME}$ otherwise.

```
n = int(input())
prime= True
```

```
for i in range(2,n):
    if(n%i==0):
        prime= False
if prime:
    print('PRIME')
else:
    print('NOTPRIME')
```

PPA 5

Accept a sequence of positive integers as input and print the the maximum number in the sequence. The input will have $n + 1$ lines, where n denotes the number of terms in the sequence. The i^{th} line in the input will contain the i^{th} term in the sequence for $1 <= i<= n$. The last line of the input will always be the number 0. Each test case will have at least one term in the sequence.

```
n = int(input())
max = 0
while(n!=0):
  if max < n:
    max = n
  n = int(input())
print(max)
```

PPA 6

Accept a sequence of words as input and print the the shortest word in the sequence. The input will have $n + 1$ lines, where n denotes the number of terms in the sequence. The i^{th} line in the input will contain the i^{th} word in the sequence for $1 <= i<= n$. The last line of the input will always be the string $abcdefghijklmnopqrstuvwxyz$. This string is not a part of the sequence. You can assume that each test case corresponds to a non-empty sequence of words. If there are multiple words that have the same minimum length, print the first such occurrence.

```
s = input()
small = s
while (s!= 'abcdefghijklmnopqrstuvwxyz'):
  if(len(s))<len(small):
    small = s
  s = input()
print(small)
```

PPA 7

Accept a positive integer as input and print the sum of the digits in the number.

```
n = input()
sum = 0
for i in n:
    sum += int(i)
print (sum)
```

PPA 8

Accept a positive integer n as input and print the first n integers on a line separated by a comma.

```
n = int(input())
for i in range (1,n):
print(i, end=',')
print(n)
```

PPA 9

```
n = int(input())
for i in range(1,n+1):
  for j in range(1,i+1):
print(0,end = '')
print()
```

PPA 10

Accept a positive integer n as input and print the sum of all prime numbers in the range $[1, n]$, endpoints inclusive. If there are no prime numbers in the given range, then print 0.

```
n = int(input())
sum = 0
prime = True
for i in range(2,n+1):
  for j in range (2,i):
    if i%j == 0:
      prime = False
  if prime:
    sum+=i
  prime = True
print(sum)
```

PPA 11

Accept a positive integer n as input and find all solutions to the equation:

$$x^2+y^2=z^2$$

subject to the following constraints:

(1) x, y and z are positive integers

(2) $x<y<z<n$

Print each solution triplet on one line — x,y,z — with a comma between consecutive integers. The triplets should be printed in ascending order. If you do not find any solutions satisfying the given constraints, print the string NO SOLUTION as output.

Order relation among triplets

Given two triplets $T_1 = (x_1, y_1, z_1)$ and $T_2 = (x_2, y_2, z_2)$, use the following process to compare them:

(1) If $x_1 < x_2$, then $T_1 < T_2$

(2) If $x_1 = x_2$ and$y_1 < y_2$, then $T_1 < T_2$

(3) If $x_1 = x_2$ and $y_1 = y_2$ and $z_1 < z_2$, then $T_1 < T_2$

```
n=int(input())
c=True
for x in range(1,n):
    for y in range(1,n):
        for z in range(1,n):
            if (x*x)+(y*y)==z*z and x<y<z<n:
print(x,y,z, sep=',')
                c=False

if c:
print('NO SOLUTION')
```

PPA 12

Accept two strings as input and form a new string by removing all characters from the second string which are present in the first string. Print this new string as output. You can assume that all input strings will be in lower case.

```
a = input()
b = input()
for c in a:
  if c in b:
      b = b.replace(c,'')
print(b)
```

GrPA 1

Accept a positive integer n as input and print the sum of the first n terms of the series given below:

$$1 + (1 + 2) + (1 + 2 + 3) + (1 + 2 + 3 + 4) + \dots$$

Just to be clear, the first term in the series is 1, the second term is $(1 + 2)$ and so on.

```
n = int(input())
total = 0
for i in range(1, n + 1):
    for j in range(1, i + 1):
        total = total + j
print(total)
```

GrPA 2

Accept a positive integer n, with $n > 1$, as input from the user and print all the prime factors of n in ascending order.

```
n = int(input())
```

```
for f in range(2, n + 1):
    # first check if f is a factor of n
    if n % f == 0:
        # now check if f is a prime
is_prime = True
        for i in range(2, f):
            if f % i == 0:
is_prime = False
                break
        if is_prime:
            print(f)
```

GrPA 3

A bot starts at the origin — (0,0) — and can make the following moves:

- UP
- DOWN
- LEFT
- RIGHT

Each move has a magnitude of 1 unit. Accept the sequence of moves made by the bot as input. The first entry in the sequence is always START while the last entry in the sequence is always STOP. A sample sequence is given below:

START

UP

RIGHT

LEFT

LEFT

DOWN

UP

STOP

Print the Manhattan distance of the bot from the origin. If the bot is at the position (x, y), then its Manhattan distance from the origin is given by the equation:

$$D = |x|+|y|$$

```
x, y = 0, 0          # start at the origin
seq = input()
while seq != 'STOP':
    if seq == 'UP':
        y += 1
    if seq == 'DOWN':
        y -= 1
    if seq == 'LEFT':
        x -= 1
    if seq == 'RIGHT':
        x += 1
seq = input()
# abs(x) is the absolute value of x
# abs(-1) is 1
dist = abs(x) + abs(y)
print(dist)
```

GrPA 4

Accept a string as input, convert it to lower case, sort the string in alphabetical order, and print the sorted string to the console. You can assume that the string will only contain letters.

```
s=input().lower()
a='abcdefghijklmnopqrstuvwxyz'
t=''
for x in a:
    for y in s:
        if x==y:
            t+=y
print(t)
```

GrPA 5

Accept a phone number as input. A valid phone number should satisfy the following constraints.

(1) The number should start with one of these digits: 6, 7, 8, 9

(2) The number should be exactly 10 digits long.

(3) No digit should appear more than 7 times in the number.

(4) No digit should appear more than 5 times in a row in the number.

If the fourth condition is not very clear, then consider this example: the number 9888888765 is invalid because the digit 8 appears more than 5 times in a row. Print the string valid if the phone number is valid. If not, print the string invalid.

```
n=input()
a=False
if len(n)==10 and int(n[0])>5 and n.isdigit():
  for i in range(10):
    if n.count(n[i])<8:
      a=True
      if n[i]*6 in n:
        a=False
        break
    else:
        break
if a:
    print('valid')
else:
    print('invalid')
```

GrPA 6

Accept a positive integer n as input and print a "number arrow" of size n. For example, $n = 5$ should produce the following output:

```
1
1,2
1,2,3
1,2,3,4
1,2,3,4,5
1,2,3,4
1,2,3
1,2
1
```

You can assume that n is greater than or equal to 2 for all test cases. Hint: range(5, 0, -1) is the sequence 5, 4, 3, 2, 1

```
n=int(input())
for i in range(1,n+2):
  for j in range(1,i):
    if j<i-1:
      print(j,end=',')
    else:
      print(j)
for i in range(n,0,-1):
  for j in range(1,i):
    if j<i-1:
      print(j,end=',')
    else:
      print(j)
```

Week 4

PPA 1

Accept a positive integer n as input and print the list of first n positive integers as output.

```
n = int(input())
l=[]
for i in range(1,n+1):
l.append(i)
print(l)
```

PPA 2

Accept a sequence of words as input, append all these words to a list in the order in which they are entered, and print this list as output. The first line in the input is a positive integer n that denotes the number of words in the sequence. The next n lines will have one word on each line.

```
n = int(input())
l=[]
for i in range(n):
    x = input()
l.append(x)
print(l)
```

PPA 3

Accept a sequence of comma-separated integers as input and print the maximum value in the sequence as output.

Hint:
When in doubt, always print the variables and examine the output.

```
num='1,2,3,4,5'
L=num.split(',')
```

```
num = input().split(',')
max = -1
for i in range(len(num)):
    if(int(num[i])>max):
        max = int(num[i])
```

```
print(max)
```

PPA 4

This question introduces you to the idea of prefix codes. **Prefix code** is a block of visible code that is already provided to you. You have to type your code *below* the prefix code. Note that the contents of the prefix *cannot* be modified.

A list `L` of words is already given to you as a part of the prefix code. Print the longest word in the list. If there are multiple words with the same maximum length, print the one which appears at the rightmost end of the list.

You do not have to accept input from the console as it has already been provided to you

```
L = input().split(',')
max=0
m=''
for i in range(len(L)):
  if len(L[i])>=max:
    max=len(L[i])
    m=''
    m+=L[i]
print(m)
```

PPA 5

Accept a space-separated sequence of positive real numbers as input. Convert each element of the sequence into the greatest integer less than or equal to it. Print this sequence of integers as output, with a comma between consecutive integers.

```
x=input()
r=x.split(' ')
i = 0
s=''
while i<( len(r)-1) :
  n = int(float(r[i]))
  s += str(n)+','
i += 1
n= int(float(r[i]))
s += str(n)
print(s)
```

PPA 6

Accept a sequence of comma-separated words as input. Reverse the sequence and print it as output.

Hint:

```
print([1]+[2])
print([2]+[1])
```

```
l=input().split(',')
for i in range(len(l)-1,0,-1):
    print(l[i],end=',')
print(l[0])
```

PPA 7

This question introduces you to the idea of suffix codes. **Suffix code** is a block of visible code that will be executed after whatever code you type. You have to type your code *above* the suffix code. Note that the contents of the suffix code *cannot* be modified.

Accept a square matrix as input and store it in a variable named `matrix`. The first line of input will be, n, the number of rows in the matrix. Each of the next n lines will have a sequence of nspace-separated integers.

You do not have to print the output to the console as the suffix code already does that for you.

```
matrix=[]
n=int(input())
for i in range(n):
    f=input().split(' ')
    for j in range(n):
        f[j]=int(f[j])
matrix.append(f)
print(matrix)
```

PPA 8

An identity matrix is a square matrix which has ones on the main diagonal and zeros everywhere else. For example, the identity matrix of size 3×3 is:

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Accept a positive integer n as input and print the identity matrix of size *n*×*n*. Your output should have nlines, where each line is a sequence of n comma-separated integers that corresponds to one row of the matrix.

```
n=int(input())
for i in range(0,n):
    for j in range(0,n):
        if(i==j):
            if(j==(n-1)):
print('1')
            else:
                print('1',end=',')
        else:
            if(j==(n-1)):
```

```
print('0')
            else:
                print('0',end=',')
```

PPA 9

Accept a square matrix A and an integer s as input and print the matrix *s*·*A* as output. Multiplying a matrix by an integer s*s* is equivalent to multiplying each element of the matrix by s. For example,

$$2\cdot\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}=\begin{bmatrix}2 & 4\\ 6 & 8\end{bmatrix}$$

The first line of input is a positive integer, n, that denotes the dimension of the matrix A. Each of the next n lines contains a sequence of space-separated integers. The last line of the input contains the integer s.
Print the matrix *s*·*A* as output. Each row of the matrix must be printed as a sequence of space separated integers, one row on each line.

```
m=[]
n=int(input())
for i in range(n):
    f=input().split(' ')
    for j in range(n):
        f[j]=int(f[j])
m.append(f)
s=int(input())
for i in range(n):
    for j in range(n):
        m[i][j]=m[i][j]*s
        if j<(n-1):
            print(m[i][j],end=' ')
        else:
            print(m[i][j])
```

PPA 10

Accept two square matrices A and B of dimensions *n*×*n* as input and compute their sum A + B.

The first line will contain the integer n. This is followed by 2n lines. Each of the first n lines is a sequence of comma-separated integers that denotes one row of the matrix A. Each of the last n lines is a sequence of comma-separated integers that denotes one row of the matrix B.
Your output should again be a sequence of n lines, where each line is a sequence of comma-separated integer that denotes a row of the matrix A + B.

```
n=int(input())
a=[]
b=[]
for i in range(2):
    for j in range(n):
        f=[]
        f=input().split(',')
```

```
for k in range(n):
          f[k]=int(f[k])
        if i==0:
a.append(f)
        else:
b.append(f)
for i in range(n):
  for j in range(n):
    a[i][j]+=b[i][j]
    if j<n-1:
      print(a[i][j],end=',')
    else:
      print(a[i][j])
```

PPA 11

This question introduces you to two ideas that will keep repeating throughout this course:

- Entering your code within a function. We will cover functions next week. The only thing you need to do for this problem is to indent all your code to the right by one unit (four spaces), and paste this between the prefix and suffix code.
- The idea of invisible codes. Invisible code is a block of code that will be hidden from your sight. The invisible code will modify the code that you write. But the details of the modification will not be revealed to you.

`L` is a list of real numbers that is already given to you. You have to sort this list in descending order and store the sorted list in a variable called `sorted_L`.

You do not have to accept input from the console as it has already been provided to you. You do not have to print the output to the console. Input-Output is the responsibility of the invisible code for this problem.

```
def solution(L):
    ### Enter your solution below this line
    ### Indent your entire code by one unit (4 spaces) to the right
    for i in range (len(L)):
        for j in range(i + 1, len(L)):
            if(L[i] < L[j]):
                t = L[i]
                L[i] = L[j]
                L[j] = t
sorted_L=L
    ### Enter your solution above this line
    return sorted_L
```

GrPA 1

In the first line of input, accept a sequence of space-separated words. In the second line of input, accept a single word. If this word is not present in the sequence, print `NO`. If this word is present in the sequence, then print `YES` and in the next line of the output, print the number of times the word appears in the sequence.

```
s=input().split(' ')
a=input()
if a in s:
  print('YES')
  print(s.count(a))
else:
  print('NO')
```

GrPA 2

You are given a list `marks` that has the marks scored by a class of students in a Mathematics test. Find the median marks and store it in a float variable named `median`. You can assume that `marks` is a list of float values.

Procedure to find the median
(1) Sort the marks in ascending order. Do not try to use built-in methods. Look at the lecture 4.5 of week-4 to get a better idea.

(2) If the number of students is odd, then the median is the middle value in the sorted sequence. If the number of students is even, then the median is the arithmetic mean of the two middle values in the sorted sequence.

You do not have to accept input from the console as it has already been provided to you. You do not have to print the output to the console. Input-Output is the responsibility of the autograder for this problem. Refer PPA-11 if you are not sure how this works.

```
def solution(marks):
    ### Enter your solution below this line
    ### Indent your entire code by one unit (4 spaces) to the right
marks_sort = []
    for x in range(len(marks)):
marks_sort.append(min(marks))
marks.remove(min(marks))
    if len(marks_sort)%2 ==0:
        median = (((marks_sort[(int(len(marks_sort)//2))-1]
+marks_sort[(int(len(marks_sort)//2)+1)-1])/2))
eliflen(marks_sort)%2 != 0:
        median = (marks_sort[(((int(len(marks_sort)))+1)//2)-1])
    ### Enter your solution above this line
    return median
```

GrPA3

Accept two square matrices A and B of dimensions $n \times n$ as input and compute their product AB.

The first line of the input will contain the integer n. This is followed by 2n lines. Out of these, each of the first n lines is a sequence of comma-separated integers that denotes one row of the matrix A. Each of the last n lines is a sequence of comma-separated integers that denotes one row of the matrix B.

Your output should again be a sequence of n lines, where each line is a sequence of comma-separated integers that denotes a row of the matrix AB.

```
n = int(input())

# Accept matrix A
A = [ ]
for i in range(n):
    row = [ ]
    for x in input().split(','):
row.append(int(x))
A.append(row)

# Accept matrix B
B = [ ]
for i in range(n):
    row = [ ]
    for x in input().split(','):
row.append(int(x))
B.append(row)

# Initialize matrix C as a zero-matrix
C = [ ]
for i in range(n):
    row = [ ]
    for j in range(n):
row.append(0)
C.append(row)

# Matrix product
for i in range(n):
    for j in range(n):
for k in range(n):
            C[i][j] += A[i][k] * B[k][j]
        if j != n - 1:
            print(C[i][j], end = ',')
        else:
            print(C[i][j])
```

GrPA 4

You are given the names and dates of birth of a group of people. Find all pairs of members who share a common date of birth. Note that this date need not be common across all pairs. It is sufficient if both members in a pair have the same date of birth.

The first line of input is a sequence of comma-separated names. The second line of input is a sequence of comma-separated positive integers. Each integer in the sequence will be in the range [1, 365], endpoints inclusive, and stands for some day in the year.

Find all pairs of names that share a common date of birth and store them in a list called `common`. Each element of this list is itself a list, and should be of the form `[name1, name2]`, such that `name1` comes before `name2` in alphabetical order.

```
names = input().split(',')
bdays = input().split(',')
n = len(names)
for i in range(n):
bdays[i] = int(bdays[i])

common = [ ]
for i in range(n):
    for j in range(n):
        if ((i != j) and
            (bdays[i] == bdays[j]) and
            names[i] < names[j]):
            pair = [names[i], names[j]]
common.append(pair)
```

GrPA 5

You are given a sequence of n points, (x_i,y_i), $1 \le i \le n$, in the 2-D plane as input. Also, you are given a point P with coordinates (x,y). Print all points in the sequence that are nearest to P. If multiple points have the same least distance from P, print the points in the order of their appearance in the sequence.

The first line of the input is an integer n, representing the number of points in the sequence. Each of the next n lines contains the co-ordinates of a point separated by comma. The last line contains the x and y co-ordinates of the point P. Assume that all the x and y co-ordinates are integers.

The distance between two points (x_1,y_1) and (x_2,y_2) is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. You can assume that the maximum distance from P to any point will not exceed 1000.

```
n = int(input())
L = [ ]
# Append all points in the sequence to the list L
for i in range(n):
L.append(input())
# Point P
point = input().split(',')
x = int(point[0])
y = int(point[1])

# List to maintain all nearest points
min_list = []
```

```
min_dist = 1000
for i in range(n):
    # One of the points in the sequence
    temp = L[i].split(',')
    # Extract the x and y coordinates of this point
temp_x = int(temp[0])
temp_y = int(temp[1])
    # Compute the distance
dist = ((x - temp_x) ** 2 + (y - temp_y) ** 2) ** 0.5
    # Check if it is the minimum distance
    if (dist<min_dist):
min_dist = dist
min_list = [L[i]]
elifdist == min_dist:
min_list.append(L[i])
for point in min_list:
    print(point)
```

Week 5

PPA 1

<u>Type</u>: single argument, single return value
The factorial of a positive integer n is the product of the first n positive integers.

Write a function named `factorial` that accepts an integer `n` as argument. It should return the factorial of `n` if `n` is a positive integer. It should return -1 if `n` is a negative integer, and it should return 1if `n` is zero.

```
deffactorial(n):
'''
    Argument:
      n: integer
    Return:
      result: integer
  '''

```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def factorial(n):
    if n<0:
        return -1
elif n==0:
        return 1
    else:
        f=1
        for i in range(1,n+1):
            f*=i
        return f
```

PPA 2

In the Gregorian calendar, a leap year has a total of 366 days instead of the usual 365 as a result of adding an extra day (February 29) to the year. This calendar was introduced in 1582 to replace the flawed Julian Calendar. The criteria given below are used to determine if a year is a leap year or not.

- If a year is divisible by 100 then it will be a leap year if it is also divisible by 400.
- If a year is not divisible by 100, then it will be a leap year if it is divisible by 4.

Write a function named `check_leap_year` that accepts a year between 1600 and 9999 as argument. It should return `True` if the year is a leap year and `False` otherwise.

```
defcheck_leap_year(year):
'''
    Argument:
      year: integer
    Return:
is_leap_year: bool
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def check_leap_year(year):
    if year%100==0:
        if year%400==0:
            return True
        else:
            return False
    else:
        if year%4==0:
            return True
        else:
            return False
```

PPA 3

Type: multiple arguments, single return value
Write a function named `maxval` that accepts three integers `a`, `b` and `c` as arguments. It should return the maximum among the three numbers.

```
defmaxval(a,b,c):
'''
    Arguments:
      a, b, c: integers
    Return:
max_of_three: integer
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def maxval(a, b, c):
    max=a
    if b>max:
        max=b
    if c>max:
        max=c
    return max
```

PPA 4

Write a function named `dim_equal` that accepts two matrices `A` and `B` as arguments. It should return `True` if the the dimensions of both matrices are the same, and `False` otherwise.

```
defdim_equal(A,B):
'''
    Arguments:
      A, B: list of lists
    Return:
      result: bool
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def dim_equal(A, B):
    if len(A)==len(B) and len(A[0])==len(B[0]):
        return True
    else:
        return False
```

PPA 5

<u>Type</u>: single argument, multiple return values
Write a function named `first_three` that accepts a list `L` of distinct integers as argument. It should return the first maximum, second maximum and third maximum in the list, in this order. You can assume that the input list will have a size of at least three. What concept in CT does this remind you of?

```
deffirst_three(L):
'''
```

```
    Argument:
        L: list
    Return:
        fmax, smax, tmax: three integers
'''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def first_three(L):
    fmax=-10000
smax=-10000
tmax=-10000
    for i in range(len(L)):
        if L[i]>fmax:
fmax,smax,tmax=L[i],fmax,smax
elif L[i]>smax:
smax,tmax=L[i],smax
elif L[i]>tmax:
tmax=L[i]
    return fmax,smax,tmax
```

PPA 6

Function Calls

A class of English words is called *mysterious* if it satisfies certain conditions. These conditions are hidden from you. Instead, you are given a function named `mysterious` that accepts a word as argument and returns `True` if the word is mysterious and `False` otherwise.

Write a function named `type_of_sequence` that accepts a list of words as an argument. Its return value is a string that depends on the number of mysterious words in the sequence. The exact conditions are given in the following table. If k denotes the number of mysterious words in the sequence, then:

k	Return value
Less than 2	mildly mysterious
Greater than or equal to 2 but less than 5	moderately mysterious
Greater than or equal to 5	most mysterious

```
deftype_of_sequence(L):
'''
    Argument:
        L: list of strings
    Return:
seq_type: string
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def type_of_sequence(L):
    k=0
    for i in range(len(L)):
        if mysterious(L[i]):
            k+=1
    if k<2:
        return 'mildly mysterious'
elif k<5:
        return 'moderately mysterious'
    else:
        return 'most mysterious'
```

PPA 7

In a throwback to CT days, write the definition of the following five functions, all of which accept a list `L` as argument.
(1) `is_empty`: return `True` if the list is empty, and `False` otherwise.
(2) `first`: return the first element if the list is non-empty, return `None` otherwise.
(3) `last`: return the last element if the list is non-empty, return `None` otherwise.
(4) `init`: return the first n - 1$n-1$ elements if the list is non-empty and has size nn, return `None` otherwise. Note that if `L` has just one element, `init(L)` should return the empty list.
(5) `rest`: return the last n - 1$n-1$ elements if the list is non-empty and has size nn, return `None` otherwise. Note that if `L` has just one element, `rest(L)` should return the empty list.

You do not have to accept input from the user or print output to the console. You just have to write the definition of all the five functions. Each test case corresponds to one function call.

```
def is_empty(l):
    if len(l)==0:
        return True
    else:
        return False

def first(l):
    if not is_empty(l):
        return l[0]
    else:
        return 'None'

def last(l):
    if not is_empty(l):
        return l[-1]
    else:
        return 'None'

def init(l):
    if not is_empty(l):
        return l[:-1]
    else:
        return 'None'

def rest(l):
    if not is_empty(l):
        return l[1:]
    else:
        return 'None'
```

PPA 8

Write a recursive function named `fibo` that accepts a positive integer `n` as argument and returns the *nth* Fibonacci number. For this problem,$F_1=F_2=1$ are the first two Fibonacci numbers.

```
deffibo(n):
'''
    Argument:
      n: int
    Return:
f_n: int
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def fibo(n):
    if n==1 or n==2:
        return 1
    else:
        f=0
        f=fibo(n-1)+fibo(n-2)
        return f
```

PPA 9

Implement the following functions.

(1) Write a function named `get_column` that accepts a matrix named `mat` and a non-negative integer named `col` as arguments. It should return the column that is at index `col` in the matrix `mat` as a list. Zero-based indexing is used here.

(2) Write a function named `get_row` that accepts a matrix named `mat` and a non-negative integer named `row` as arguments. It should return the row that is at index `row` in the matrix `mat` as a list. Zero-based indexing is used here.

```
defget_column(mat,col):
'''
    Argument:
        mat: list of lists
        col: integer
    Return:
col_list: list
  '''
pass

defget_row(mat,row):
'''
    Argument:
        mat: list of lists
        row: integer
    Return:
row_list: list
  '''
pass
```

You do not have to accept input from the user or print output to the console. You just have to write the definition of both the functions. Each test case will correspond to one function call.

```
def get_column(mat, col):
    l=[]
    for i in range(len(mat)):
l.append(mat[i][col])
    return l
def get_row(mat, row):
    l=[]
    for i in range(len(mat[0])):
l.append(mat[row][i])
    return l
```

PPA 10

Write a function named `insert` that accepts a *sorted* list `L` of integers and an integer `x` as input. The function should return a sorted list with the element `x` inserted at the right place in the input list. The original list should not be disturbed in the process. You can assume that the input list will be sorted in ascending order.

```
definsert(L,x):
'''
    Arguments:
        L: list
        x: integer
    Return:
sorted_L: list
  '''
```

(1) The only built-in methods you are allowed to use are `append` and `remove`. You should not use any other method provided for lists.
(2) You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def insert(L, x):
L.append(x)
    for i in range(len(L)):
        for j in range(i+1,len(L)):
            if L[i]>L[j]:
                L[i],L[j]=L[j],L[i]
    return L
```

GrPA 1

The range of a list of numbers is the difference between the maximum and minimum values in the list.

Write a function named `get_range` that accepts a non-empty list of real numbers as argument. It should return the range of the list.

```
defget_range(L):
'''
    Argument:
        L: list
    Return:
        range: float
  '''
```

Note
(1) Avoid using built-in function such as `max` and `min`.
(2) You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
# get the maximum
def get_max(L):
    maxi = L[0]
    for x in L:
        if x > maxi:
            maxi = x
    return maxi

# get the minimum
def get_min(L):
    mini = L[0]
    for x in L:
        if x < mini:
            mini = x
    return mini

# get the range
def get_range(L):
    maxi = get_max(L)
    mini = get_min(L)
    return maxi - mini
```

GrPA 2

A perfect number is a positive integer that is equal to the sum of all its divisors excluding itself. For example, 6is a perfect number as 6 = 1 + 2 + 3.

Write a function named `is_perfect` that accepts a positive integer `n` as argument and returns `True` if it is a perfect number, and `False` otherwise.

```
defis_perfect(n):
'''
    Argument:
        n: int
    Return:
        result: bool
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def is_perfect(num):
    # Factor sum
fsum = 0
    for f in range(1, num):
        if num % f == 0:
fsum += f
    # fsum == num is a Boolean expression
    # It will evaluate to True if num is a perfect number
    # And False otherwise
    return fsum == num
print(is_perfect(int(input())))
```

GrPA 3

The distance between two different letters in the English alphabet is defined as one more than the number of letters between them. Alternatively, it can be defined as the number of steps needed to move from the alphabetically smaller letter to the larger letter. This is always a non-negative integer. The distance between any letter and itself is always zero. For example:

Letter-1	Letter-2	Distance
a	a	$d_{\text{letter}}(a,a)=0$
a	c	$d_{\text{letter}}(a,c)=2$
a	z	$d_{\text{letter}}(a,z)=25$
z	a	$d_{\text{letter}}(z,a)=25$
e	a	$d_{\text{letter}}(e,a)=4$

The distance between two words is defined as follows:

- It is -1, if the words are of unequal lengths.
- If the word-lengths are equal, it is the sum of the distances between letters at corresponding positions in the words. For example:

$d_{\text{word}}(\text{dog},\text{cat})=d_{\text{letter}}(\text{d},\text{c})+d_{\text{letter}}(\text{o},\text{a})+d_{\text{letter}}(\text{g},\text{t})=1+14+13=28$

Write a function named `distance` that accepts two words as arguments and returns the distance between them.

```
def distance(word_1,word_2):
'''
    Arguments:
        word_1, word_2: strings
    Return:
word_distance: int
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def distance(word_1, word_2):
    # first condition
    if len(word_1) != len(word_2):
        return -1
    letters = 'abcdefghijklmnopqrstuvwxyz'
    size = len(word_1)
dist = 0
    for i in range(size):
        c1, c2 = word_1[i], word_2[i]
        # distance between letters
        d = abs(letters.index(c1) - letters.index(c2))
dist += d
    return dist
```

GrPA 4

A $n \times n$ square matrix of positive integers is called a magic square if the following sums are equal:

- row-sum: sum of numbers in every row; there are n such values, one for each row
- column-sum: sum of numbers in every column; there are n such values, one for each column
- diagonal-sum: sum of numbers in both the main diagonals; there are two values

There are n + n + 2 = 2n + 2 values involved. All these values must be the same for the matrix to be a magic-square.

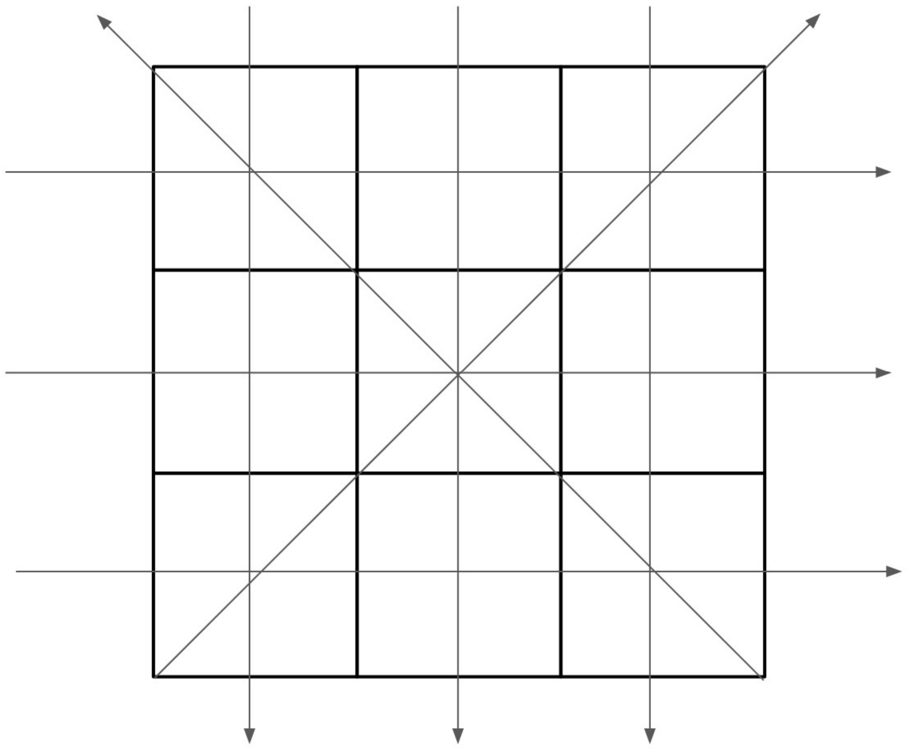
Write a function named `is_magic` that accepts a square matrix as argument and returns `YES` if it is a magic-square and `NO` if it isn't one.

```
defis_magic(mat):
'''
    Argument:
        mat: list of lists
    Return:
        string: 'YES' or 'NO'
  '''
```

Notes
(1) The cells of a magic square need not be distinct. Some or even all the cells could be identical.

(2) You do not have to accept input from the user or print output to the console. You just have to write the function definition.

A sample-image for a 3X3 matrix that details the various sums needed. Note that the input need not be restricted to 3X3matrices:



```
def is_magic(mat):
    # first get the dimension of the matrix
    m = len(mat)
    # the sum of the two diagonals
    d1sum, d2sum = 0, 0
    # (i, i) goes from top-left -> bottom-right
    # (i, m - i - 1) goes from top-right -> bottom-left
    # note that a single loop is enough; no nesting required
    for i in range(m):
        d1sum += mat[i][i]
        d2sum += mat[i][m - i - 1]
    # if the two diagonal sums are unequal, we can return NO
  # unnecessary computation can be avoided
    if not(d1sum == d2sum):
        return 'NO'
    # get row-sum and column-sum
    for i in range(m):
rsum, csum = 0, 0
        for j in range(m):
rsum += mat[i][j]
csum += mat[j][i]
        if not(rsum == csum == d1sum):
            return 'NO'
    # if the code reaches this level
    # then all requirements of a magic-square are satisfied
    # so we can safely return YES
    return 'YES'
```

GrPA5

The transpose of a matrix is obtained by swapping its rows and columns:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \rightarrow \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}$$

Write a function named `transpose` that accepts a matrix `mat` as input and returns its transpose.

```
deftranspose(mat):
'''
    Argument:
        mat: list of lists
    Return:
mat_trans: list of lists
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
# Refer PPAs to understand this function
def get_column(mat, col):
col_list = [ ]
    m = len(mat)
    for row in range(m):
col_list.append(mat[row][col])
    return col_list

# We make use of get_column
# to get the ith column
# this is made the ith row in the transpose
def transpose(mat):
    m, n = len(mat), len(mat[0])
mat_trans = [ ]
    for i in range(n):
mat_trans.append(get_column(mat, i))
    return mat_trans
```

Week 6

PPA 1

Accept a sequence of words as input. Create a dictionary named `freq` whose keys are the distinct words in the sequence. The value corresponding to a key (word) should be the frequency of occurrence of the key (word) in the sequence.

---

(1) You can assume that all words will be in lower case.

(2) You do not have to print the output to the console. This will be the responsibility of the autograder.

```
freq = dict()
L = input().split(',')

for word in L:
freq[word] = 0

for word in L:
freq[word] = freq[word] + 1
```

PPA 2

Accept a positive integer as input and print the digits present in it from left to right. Each digit should be printed as a lower case word on a separate line. How would you use dictionaries to solve this problem?

```
num = input()

D = {'0': 'zero', '1': 'one', '2': 'two', '3': 'three', '4': 'four',
     '5': 'five', '6': 'six', '7': 'seven', '8': 'eight', '9': 'nine'}

for digit in num:
    print(D[digit])
```

PPA 3

Write the following functions:

(1) `is_key`: accept a dictionary `D` and a variable `key` as arguments. Return `True` if the variable `key` is a key of the dictionary `D`, and `False` otherwise.
(2) `value`: accept a dictionary `D` and a variable `key` as arguments. If the variable `key` is not a key of the dictionary `D`, return `None`, otherwise, return the value corresponding to this `key`.

```
defis_key(D,key):
'''
    Arguments:
        D: dict
        key: could be of any type
    Return:
        bool
  '''
pass

defvalue(D,key):
'''
    Arguments:
        D: dict
        key: could be of any type
    Return:
        result: depends on the dict; refer problem statement
  '''

```

You do not have to accept input from the user or print the output to the console. You just have to write the definition of both the functions.

```
def is_key(D, key):
    return key in D

def value(D, key):
    if is_key(D, key):
        return D[key]
    else:
        return None
```

PPA 4

Write a function named `value_to_keys` that accepts a dictionary `D` and a variable named `value` as arguments. It should return the list of all keys in the dictionary that have value equal to `value`. If the value is not present in the dictionary, the function should return the empty list.

```
defvalue_to_keys(D, value):
'''
    Arguments:
        value: could be of any type
    Return:
        keys: list
  '''

```

(1) You do not have to accept input from the user or print the output to the console. You just have to write the definition of the function.

(2) The keys inside the list could be in any order.

```
def value_to_keys(D, value):
    values = [ ]
    for key in D:
        if D[key] == value:
values.append(key)
    return values
```

PPA 5

Write the following functions:

(1) `dict_to_list`: accept a dictionary `D` as argument. Return the key-value pairs in `D` as a list `L` of tuples. That is, every element of `L` should be of the form `(key, value)` such that D[key] = value. Going the other way, every key-value pair in the dictionary should be present as a tuple in the list `L`.
(2) `list_to_dict`: accept a list of tuples `L` as argument. Each element of `L` is of the form `(x, y)`. Return a dict `D` such that each tuple `(x, y)` corresponds to a key-value pair in `D`. That is, D[x] = y.

```
defdict_to_list(D):
'''
    Argument:
        D: dict
    Return:
        L: list of tuples
  '''
pass

deflist_to_dict(L):
'''
    Argument:
        L: list of tuples
    Return:
        D: dict
  '''
pass
```

---

(1) For the function `dict_to_list(D)`, the order in which the key-value pairs are appended to the list doesn't matter.
(2) For the function `list_to_dict(L)`, you can assume that if `(x1, y1)` and `(x2, y2)` are two different elements in `L`, `x1 != x2`. Why is this assumption important?
(3) You do not have to accept input from the user or print the output to the console. You just have to write the definition of both the functions.

```
def dict_to_list(D):
    L = [ ]

    for key in D:
L.append((key, D[key]))
```

```
    return L

def list_to_dict(L):
    D = dict()

    for key, value in L:
        D[key] = value

    return D
```

PPA 6

Scores Dataset Revisited

Recall the Scores dataset from CT. We shall be using a variant of this dataset for this problem. Each student-entry in the dataset is represented as a dictionary. For example, one of the entries would look like this:

```
{'SeqNo':1,'Name':'Devika','Gender':'F','City':'Bengaluru',
'Mathematics':85,'Physics':100,'Chemistry':79,'Biology':75,
'Computer Science':88,'History':60,'Civics':88,'Philosophy':95}
```

All keys of the dict are strings. For `SeqNo` and all the subjects, the corresponding values are integers. The values corresponding to `Name`, `Gender` and `City` are strings.
The entire dataset is represented as a list of dictionaries. That is, each element of the list will be a dictionary like the one given above. This list is named `scores_dataset`. The `SeqNo` is a unique identifier for each student that runs from 00 to n - 1, where n is the total number of students in the dataset.

Write a function named `get_marks` that accepts the `scores_dataset` and a variable named `subject` as arguments. It should return the marks scored by all students in `subject` as a list of tuples. Each element in this list is of the form `(Name, Marks)`. The order in which the tuples are appended to the list doesn't matter.

```
defget_marks(scores_dataset,subject):
'''
    Arguments:
scores_dataset: list of dicts
      subject: string
    Return:
      marks: list of tuples of the form (string, int)
  '''
```

---

(1) You do not have to accept input from the user or print the output to the console. You just have to write the definition of the function.

(2) Do not try to process the output produced. We randomly sample five elements from the list returned by your function and print that in the desired form.

```
def get_marks(scores_dataset, subject):
    L = [ ]
    for student in scores_dataset:
        marks = student[subject]
        name = student['Name']
L.append((name, marks))
    return L
```

PPA 7

In this problem, we shall try to create the list of dictionaries which was given to us in the previous problem.

Accept a positive integer n that represents the number of students in the class. n blocks of input follow. Each block is made up of six lines and contains the details of one student in the class. Create a dictionary corresponding to each student. All keys should be strings. The type of the value corresponding to a key and the order in which the inputs should be accepted are shown in the table given below.

Line number	Key	Type of Value
1	Name	String
2	City	String
3	SeqNo	Integer
4	Mathematics	Integer
5	Physics	Integer
6	Chemistry	Integer

Append each dictionary to a list named `scores_dataset`. This is the list that we will finally use for evaluating your code. The dictionaries corresponding to the students should be appended in the order in which they appear in the sequence of inputs.

You do not have to print the output to the console.

```
n = int(input())

scores_dataset = [ ]

for i in range(n):
    record = dict()
    record['Name'] = input()
    record['City'] = input()
    record['SeqNo'] = int(input())
    record['Mathematics'] = int(input())
    record['Physics'] = int(input())
    record['Chemistry'] = int(input())
```

```
scores_dataset.append(record)
```

PPA 8

Write the following functions:

(1) `factors`: accept a positive integer nn as argument. Return the set of all factors of nn.
(2) `common_factors`: accept two positive integers aa and bb as arguments. Return the set of common factors of the two numbers. This function must make use of `factors`.
(3) `factors_upto`: accept a positive integer nn as argument. Return a dict `D`, whose keys are integers and values are sets. Each integer in the range [1, n], endpoints inclusive, is a key of `D`. The value corresponding to a `key`, is the set of all factors of `key`. This function must make use of `factors`.
The idea we are trying to bring out here is to make use of pre-defined functions whenever needed.

```
deffactors(n):
'''
    Argument:
      n: integer
    Return:
factors_of_n: set
  '''
pass

defcommon_factors(a,b):
'''
    Arguments:
      a, b: integers
    Return:
factors_common: set
  '''
pass

deffactors_upto(n):
'''
    Argument:
      n: integer
    Return:
      result: dict (keys: integers, values: sets)
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the definition of all three functions. Each test case will correspond to one function call.

```
def factors(n):
    F = set()
    for i in range(1, n + 1):
        if n % i == 0:
F.add(i)
```

```
    return F

def common_factors(a, b):
    fa = factors(a)
    fb = factors(b)
    return fa.intersection(fb)

def factors_upto(n):
    D = dict()
    for i in range(1, n + 1):
        D[i] = factors(i)
    return D
```

PPA 9

Accept a sequence of words as input. Create a dictionary named `real_dict` whose keys are the letters of the English alphabet. For each key (letter), the corresponding value should be a list of words that begin with this key (letter). For any given key, the words should be appended to the corresponding list in the order in which they appear in the sequence. You can assume that all words of the sequence will be in lower case.

You do not have to print the output to the console.

```
L = input().split(',')

real_dict = dict()

for word in L:
    start = word[0]
    if start not in real_dict:
real_dict[start] = [ ]
real_dict[start].append(word)
```

PPA 10

The scores dataset is a list of dictionaries one of whose entries is given below for your reference:

```
{'SeqNo':1,'Name':'Devika','Gender':'F','City':'Bengaluru',
'Mathematics':85,'Physics':100,'Chemistry':79,'Biology':75,
'Computer Science':88,'History':60,'Civics':88,'Philosophy':95}
```

Write the following functions:

(1) `group_by_city`: accepts the `scores_dataset` as argument. It should return a dictionary named `cities` whose keys are names of the cities that the students are from. The value corresponding to a key (city) is the list of names of all students who hail from this city. The order in which names are appended to the list doesn't matter.
(2) `busy_cities`: accepts the `scores_dataset` as argument. It should return a list of cities. Each city in this list has the property that the number of students from this city is greater than or equal

to the number of students from every other city in the dataset. Your function should make use of `group_by_city`. The order in which the cities appended to the list doesn't matter.

```
defgroup_by_city(scores_dataset):
'''
    Argument:
scores_dataset: list of dicts
    Return:
      cities: dict: (key: string, value: list of strings)
  '''

defbusy_cities(scores_dataset):
'''
    Argument:
scores_dataset: list of dicts
    Return:
      result: list of strings
  '''

```

---

(1) You do not have to accept input from the user or print the output to the console. You just have to write the definition of both the functions.

(2) Do not try to process the output produced. We randomly sample a few elements from the dictionary or list returned by your function and print that in the desired form.

```
def group_by_city(scores_dataset):
    cities = dict()

    for student in scores_dataset:
        city = student['City']
        name = student['Name']
        if city not in cities:
            cities[city] = [ ]
        cities[city].append(name)

    return cities

def busy_cities(scores_dataset):
    cities = group_by_city(scores_dataset)

    busy = [ ]
maxpop = 0
    for city in cities:
        if len(cities[city]) >maxpop:
maxpop = len(cities[city])
            busy = [city]
eliflen(cities[city]) == maxpop:
busy.append(city)
```

```
    return busy
```

GrPA 1

The scores dataset is a list of dictionaries one of which is given below for your reference:

```
{'SeqNo':1,'Name':'Devika','Gender':'F','City':'Bengaluru',
'Mathematics': 94,'Physics':84,'Chemistry':79,'Biology':99,
'Computer Science':88,'History':63,'Civics':88,'Philosophy':85}
```

Write a function named `get_toppers` that accepts three arguments in this order:

- `scores_dataset`
- `subject`
- `gender`

It should a return a list of the names of students who belong to the gender given by the argument `gender` ('F' or 'M') and have topped in the subject given by the argument `subject`. As there could be multiple toppers, the function should return a list of names.

```
defget_toppers(scores_dataset,subject,gender):
'''
     Arguments:
scores_dataset: list of dicts
        subject: string
        gender: string ('F' or 'M')
     Return:
        toppers: list of strings
  '''
```

(1) The names could be appended to the list in any order.

(2) Just to be clear, a topper is a student who has scored the maximum marks in the subject.
(3) You do not have to accept input from the user or print the output to the console. You just have to write the definition of the function.

```
# Refer PPA-6 of week-6 to understand how get_marks
def get_marks(scores_dataset, subject, gender):
    L = [ ]
    for student in scores_dataset:
        if student['Gender'] == gender:
            marks = student[subject]
            name = student['Name']
L.append((name, marks))
    return L

def get_toppers(scores_dataset, subject, gender):
    # get the list of tuples
    L = get_marks(scores_dataset, subject, gender)
    toppers = [ ]
maxmarks = 0
    for i in range(len(L)):
        # L[i][0] -> name, L[i][1] -> marks
        if L[i][1] >maxmarks:
maxmarks = L[i][1]
            # if a new max is found,
            # create a new list and add L[i][0] as first element
            toppers = [L[i][0]]
        # if two have obtained same maximum, just append L[i][0] to current
list
elif L[i][1] == maxmarks:
toppers.append(L[i][0])
    return toppers
```

GrPA 2

Write a function named `freq_to_words` that accepts a list of words as argument. It should return a dictionary which has the following structure:

- key: frequency of words in the list
- value: list of all words that have the above frequency

```
deffreq_to_words(words):
'''
    Argument
      words: list of strings
    Return:
      result: dictionary
        key: integer
        value: list of strings
  '''
```

Sample input-output behaviour:

words	freq_to_words(words)
['a', 'random', 'collection', 'a', 'another', 'a', 'random']	{1: ['another', 'collection'], 2: ['random'], 3: ['a']}
['one', 'two', 'three', 'one']	{1: ['three', 'two'], 2: ['one']}

---

(1) All words in the input list will be in lower case.

(2) The order in which the words are appended to the list doesn't matter.

(3) You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
# Refer PPA-1 of week-6 for this function
def words_to_frequency(words):
```

```
freq_dict = dict()
    for word in words:
        if word not in freq_dict:
freq_dict[word] = 0
freq_dict[word] += 1
    return freq_dict

def freq_to_words(words):
freq_dict = words_to_frequency(words)

    result = dict()
    for word in freq_dict:
freq = freq_dict[word]
        if freq not in result:
            result[freq] = [ ]
        result[freq].append(word)

    return result
```

GrPA 3

Write a function named `rotate` that accepts a matrix `mat` as argument. It should return a matrix that is rotated by 90˚ in the clockwise direction. For example:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \rightarrow \begin{bmatrix} d & a \\ e & b \\ f & c \end{bmatrix}$$

```
defrotate(mat):
'''
    Argument:
        mat: list of lists
    Return:
rotated_mat: list of lists
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def rotate(mat):
    #print(mat)
    rotated=[]
    for i in range(len(mat[0])):
        turned=[]
        for j in mat[::-1]:
            #print(j[i])
turned.append(j[i])
rotated.append(turned)
    return rotated
```

GrPA 4

Write a function named `two_level_sort` that accepts a list of tuples named `scores` as argument. Each element in this list is of the form `(Name, Marks)` and represents the marks scored by a student in a test: the first element is the student's name and the second element is his or her marks.
The function should return a list of tuples that is sorted in two levels:

- Level-1: ascending order of marks
- Level-2: alphabetical order of names among those students who have scored equal marks

Each element in the returned list should also be of the form `(Name, marks)`. Note that level-2 should not override level-1. That is, after the second level of sorting, the list should still be sorted in ascending order of marks. Additionally, the students having the same marks should appear in alphabetical order.
Sample input-output behaviour

scores	two_level_sort(scores)
[('Harish', 80), ('Aparna', 90), ('Harshita', 80)]	[('Harish', 80), ('Harshita', 80), ('Aparna', 90)]
[('Sachin', 85), ('Yuvan', 65), ('Anita', 85)]	[('Yuvan', 65), ('Anita', 85), ('Sachin', 85)]

```
deftwo_level_sort(scores):
'''
    Argument:
      scores: list of tuples, (string, integer)
    Return:
      result: list of tuples (string, integer)
  '''
```

(1) You should not use any built-in sort functions to solve this problem.

(2) You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def two_level_sort(scores):
sorted_L = [ ]

    while scores != [ ]:
        # minentry[0] -> name, minentry[1] -> score
minentry = scores[0]
        for i in range(len(scores)):
            # scores[i][0] -> name, scores[i][1] -> marks
            if scores[i][1] <minentry[1]:
minentry = scores[i]
            # If scores are equal, check for alphabetical order
elif (scores[i][1] == minentry[1] and
                  scores[i][0] <minentry[0]):
minentry = scores[i]
```

```
sorted_L.append(minentry)
scores.remove(minentry)

    return sorted_L
```

Week 7

GrPA 1

A round-robin tournament is one in which each team competes with every other team. Consider a version of the IPL tournament in which every team plays exactly one game against every other team. All these games have a definite result and no match ends in a tie. The winning team in each match is awarded one point.

Eight teams participate in this round-robin cricket tournament: CSK, DC, KKR, MI, PK, RR, RCB and SH. You are given the details of the outcome of the matches. Your task is to prepare the IPL points table in descending order of wins. If two teams have the same number of points, the team whose name comes first in alphabetical order must figure higher up in the table.

There are eight lines of input. Each line is a sequence of comma-separated team names. The first team across these eight lines will always be in this order: CSK, DC, KKR, MI, PK, RR, RCB and SH. For a given sequence, all the other terms represent the teams that have lost to the first team. For example, the first line of input could be: CSK,MI,DC,PK. This means that CSK has won its matches against the teams MI, DC and PK and lost its matches against all other teams. If a sequence has just one team, it means that it lost all its matches.

Print the IPL points table in the following format — team:wins — one team on each line. There shouldn't be any spaces in any of the lines.

```
results = [ ]
for i in range(8):
    L = input().split(',')
    winner = L[0]       # the first team is the winner
    losers = L[1: ]     # all these teams have lost to the winner
    # we only need the number of wins and the winning team
results.append((winner, len(losers)))

table = [ ]
# two-level-sort
# refer GrPA-4 of week-6
# we first sort by points, then by name
while results != [ ]:
maxteam = results[0]
    for i in range(len(results)):
        team = results[i]
        if team[1] >maxteam[1]:
maxteam = team
```

```
elifteam[1] == maxteam[1] and team[0] <maxteam[0]:
maxteam = team
results.remove(maxteam)
table.append(maxteam)

for team in table:
    print(f'{team[0]}:{team[1]}')
```

GrPA 2

Two dictionaries D1 and D2 can be merged to create a new dictionary D that has the following structure:

- Each key-value pair in D is present either in D1 or D2.
- Each key in D1 is also a key in D. Likewise, each in D2 is also a key in D.
- If a particular key is common to both D1 and D2, the value corresponding to this key in one of the two dictionaries is retained in D.

Write a function named merge that accepts the following arguments:

- D1: first dictionary
- D2: second dictionary
- priority: The is a string variable that denotes the priority given to common keys while merging. That is, if both D1 and D2 have a key in common, then this variable will determine which value needs to be retained. More specifically, priority can take one of these two values:
  - "first": retain the value corresponding to the common key present in the first dictionary
  - "second": retain the value corresponding to the common key present in the second dictionary

This function should return the merged dictionary.

```
defmerge(D1,D2,priority):
'''
    Arguments:
      - D1: first dictionary
      - D2: second dictionary
      - priority: string
    Returns: D; merged dictionary
  '''
```

You do not have to accept the input or print the output to the console. You just have to write the function definition.

```
def merge(D1, D2, priority):
    if priority=='first':
        for i in D2:
            if i not in D1:
                D1[i]=D2[i]
        return D1
    if priority=='second':
        for i in D1:
            if i not in D2:
               D2[i]=D1[i]
        return D2
```

GrPA 3

Given a square matrix M and two indices (i, j),*Mij* is the matrix obtained by removing the *ith* row and the *jth* column of M.

Write a function named `minor_matrix` that accepts three arguments:

- `M`: a square matrix
- `i`: a non-negative integer
- `j`: a non-negative integer

The function should return the matrix *Mij* after removing the *ith* row and the *jth* column of M. Note that we use zero-based indexing throughout. That is, if the matrix M is of dimensions $n×n$, then we have $0≤i,j≤n−1$.

```
defminor_matrix(M,i,j):
'''
    Arguments:
        M: list of lists
        i: integer
        j: integer
    Return:
M_ij: list of lists
  '''
```

---

(1) You can assume that the number of rows in M*M* will be at least 33 in each test case.
(2) You do not have to accept input from the user or print the output to the console.

```
def minor_matrix(M, i, j):
    l=[]
    for a in range(len(M)):
        t=[]
        if a==i:
            continue
        else:
            for b in range(len(M[0])):
                if b==j:
                    continue
                else:
t.append(M[a][b])
l.append(t)
    return l
```

GrPA 4

You are given certain details of the trains that stop at a station. Your task is to store these details in a nested dictionary.

The first line of input is n, the number of trains that stop at the station. n blocks of input follow. The first line in each block corresponds to the train name. The second line in each block corresponds to m, the number of compartments in the train. m lines of input follow. Each of these m lines has two values separated by a comma: name of the compartment and number of passengers in it.

Your task is to create a nested dictionary named `station_dict`. The keys of the dictionary are train names, the value corresponding to a key is another dictionary. The keys of the inner dictionary are the compartment names in this train, the values are the number of passengers in each compartment. For example:

```
{
'Mumbai Express':{
'S1':10,
'S2':20,
'S3':30
},
'Chennai Express':{
'S1':10,
'S2':20,
'S3':30
}
}
```

---

(1) The values of the compartments should be represented as integers and not as strings.
(2) You do not have to print the output to the console. Do not try to print the output that you observe in the "Expected Output". You just have to process the input and create thedictionary `station_dict`.

```
n=int(input())
station_dict={}
for i in range(n):
    m=input()
    a=int(input())
    d={}
    for i in range(a):
        s=input().split(',')
        d[s[0]]=int(s[1])
station_dict[m]=d
```

Week 8

PPA 1

Write a recursive function named `triangular` that accepts a positive integer n as argument and returns the sum of the first n positive integers.

```
deftriangular(n):
'''
    Argument:
        n: integer
    Return:
        result: integer
    '''
```

You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def triangular(n):
    if n == 1:
        return 1
    return n + triangular(n - 1)
```

PPA 2

The factorial of a positive integer n is defined as follows:
$n!=1\cdot2\cdot3\cdots n$

Write a recursive function named `factorial` that accepts a positive integer nn as argument and returns the factorial of n.

```
deffactorial(n):
'''
    Argument:
        n: integer
    Return:
        result: integer
    '''
```

You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n - 1)
```

PPA 3

Write a recursive function named `multiply` accepts two positive integers a and b as argument and returns their product. You can only use $+$ and $-$ operators. You are not allowed to use the $*$ symbol anywhere in your code!

```
defmultiply(a,b):
'''
    Arguments:
        a, b: integers
    Return:
        result: integer
    '''
```

You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def multiply(a, b):
    if b == 1:
        return a
    return a + multiply(a, b - 1)
```

PPA 4

The logarithm of a number x to the base 2 is the number of times 2 has to be multiplied with itself so get *x*, and is denoted by $\log_2(x)$. For example, $\log_2(4)=2$. Note that $\log_2(1)=0$.

Write a recursive function named `logarithm` that accepts a positive integer x*x* as argument and returns $\log_2(x)$.

```
deflogarithm(x):
'''
    Argument:
        x: integer
    Result:
        result: integer
    '''
```

(1) Each test case will be a power of 2.
(2) Use of Python's standard libraries is not allowed for this problem.

(3) You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def logarithm(x):
    if x == 1:
        return 0
    return 1 + logarithm(x // 2)
```

PPA 5

Write a recursive function named `palindrome` that accepts a string `word` as argument and returns `True` if it is a palindrome and `False` otherwise.

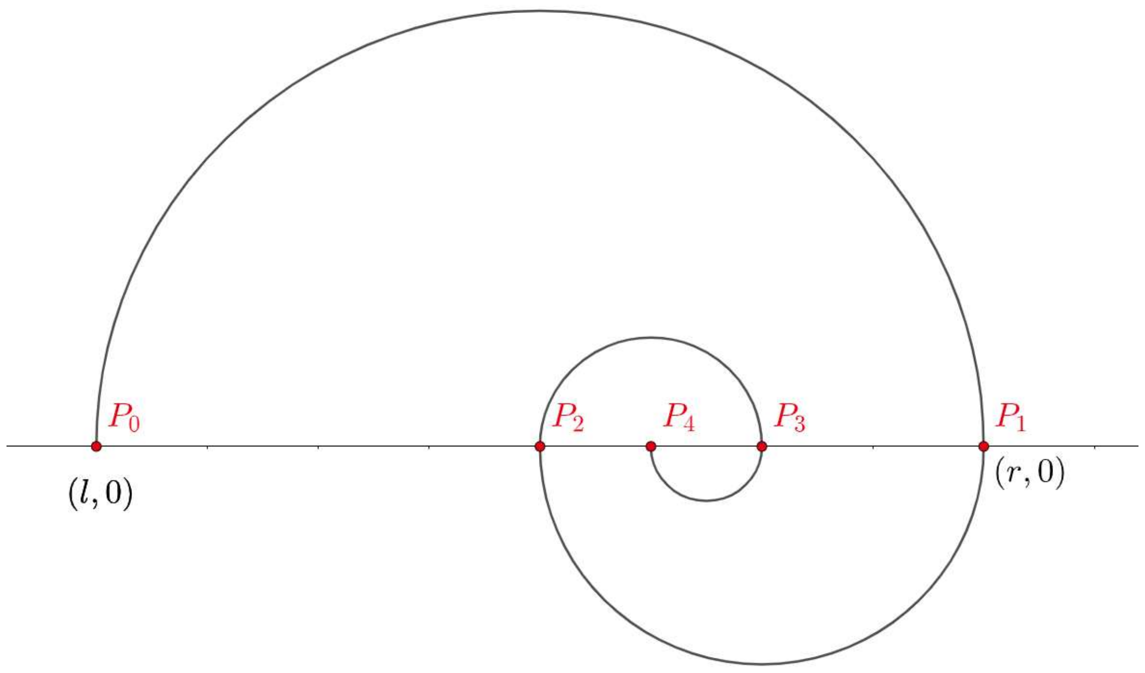
```
defpalindrome(word):
'''
    Argument:
        word: string
    Return:
        result: bool
  '''
```

You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def palindrome(word):
    if len(word) <= 1:
        return True
    if word[0] != word[-1]:
        return False
    return palindrome(word[1:-1])
```

PPA 6

Consider a spiral of semicircles. We start at a point P_0 on the x-axis with coordinates (l,0). The first arm of the spiral ends at P_1 with coordinates (r,0). The second arm of the spiral starts at P_1 and ends at the center of the first arm, P_2. The third arm starts from P_2 and ends atP_3 which happens to be the center of the second arm. And finally, the fourth arm starts at P_3 and ends at P_4, the center of the third arm.



Write two functions named `spiral_iterative` and `spiral_recursive`, each of which accepts three arguments:

- `left`: x-coordinate of the point P_0
- `right`: x-coordinate of the point P_1
- `n`: the number of arms in the spiral

Both functions should return the the x-cordinate of P_n, the point at which the *nth* arm of the spiral ends.

```
defspiral_iterative(left,right,n):
'''
    Arguments:
      left: integer
      right: integer
      n: integer
    Return:
      result: float
  '''

defspiral_recursive(left,right,n):
'''
    Arguments:
      left: integer
      right: integer
      n: integer
    Return:
      result: float
  '''
```

(1) Observe what happens as the value of n increases. Those who have taken Maths-2, can you try to answer this question without using Python, just using the concept of limits that you have learned?
(2) You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def spiral_iterative(left, right, n):
    for i in range(n - 1):
        left, right = right, (left + right) / 2
    return right


def spiral_recursive(left, right, n):
    if n == 1:
        return right
    return spiral_recursive(right, (left + right) / 2, n - 1)
```

PPA 7

Write a recursive function named `count` that accepts the following arguments:

- `L`: list of words
- `word`: a word, could be any string

This function should return the number of occurrences of `word` in `L`.

```
defcount(L,word):
'''
    Arguments:
        L: list of words
        word: string
    Return:
        result: integer
  '''
```

---

(1) You cannot use the built-in `count` method for lists in this problem.
(2) All words will be in lower case.

(3) You do not have to accept input from the user or print the output to the console. You just have to write the definition of both the functions.

```
def count(L, word):
    if len(L) == 0:
        return 0
    if L[-1] == word:
        return 1 + count(L[:-1], word)
    else:
        return count(L[:-1], word)
```

PPA 8

Write a recursive function named `non_decreasing` that accepts a non-empty list `L` of integers as argument and returns `True` if the elements are sorted in non-decreasing order from left to right, and `False` otherwise.

```
defnon_decreasing(L):
'''
    Argument:
        L: list of integers
    Return:
        result: bool
  '''
```

You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def non_decreasing(L):
    if len(L) <= 1:
        return True
    if L[-2] > L[-1]:
        return False
    return non_decreasing(L[:-1])
```

PPA 9

Write a recursive function named `uniq` that accepts a non-empty list `L` as argument and returns a new list after removing all duplicates from it. Your function must retain the last occurrence of each distinct element in the list.

```
defuniq(L):
'''
    Argument:
        L: list
    Return:
        result: list
    '''
```

You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def uniq(L):
    if len(L) == 1:
        return L
    if L[0] in L[1: ]:
        return uniq(L[1: ])
    else:
        return [L[0]] + uniq(L[1: ])
```

PPA 10

Write a recursive function named `search` that accepts the following arguments:

- `L`: a sorted list of integers
- `k`: integer

The function should return `True` if `k` is found in the list `L`, and `False` otherwise.

```
defsearch(L,k):
'''
    Arguments:
        L: sorted list of integers
        k: integer
    Return:
        result: bool
    '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def search(L, k):
    if len(L) == 0:
        return False
    if L[0] == k:
        return True
    return search(L[1: ], k)
```

PPA 11

(1) Write a function named `insert` that accepts a sorted list `L` of integers and an integer `x` as arguments. It should return a sorted list with the element `x` inserted into the input list at the right place.
(2) Write a recursive function named `isort` that accepts a non-empty list `L` of integers as argument. It should return a sorted list in ascending order. `isort` must make use of `insert`. This is a popular sorting algorithm and is called insertion sort.

```
definsert(L,x):
'''
    Arguments:
      L: list of sorted integers
      x: integer
    Return:
      result: sorted list of integers
  '''
pass
defisort(L):
'''
    Arguments:
      L: list of integers
    Return:
      result: sorted list of integers
  '''
pass

```

---

(1) Each test case corresponds to one function call.

(2) You do not have to accept input from the user or print the output to the console. You just have to write the definition of both the functions.

(3) You cannot use any built-in sort functions or methods in this problem.

```
def insert(L, x):
    if len(L) > 0:
        if x <L[0]:
            return [x] + L
        else:
            return [L[0]] + insert(L[1: ], x)
    else:
        return [x]
```

```
def isort(L):
    if len(L) == 1:
        return L
    return insert(isort(L[: -1]), L[-1])
```

PPA 12

A polynomial is a mathematical function of the following form:

$f(x)=a_0x^0+a_1x^1+a_2x^2+\cdots+a_nx^n$

The a_is are called the coefficients of the polynomial and uniquely determine it. This polynomial can be represented in Python using a list of its coefficients:

L = [a_0, a_1, a_2, . . ., a_n]

Note that L[i] corresponds to the coefficient a_i of x^i in $f(x)$, for $0\leq i\leq n$.

Write a recursive function named `poly` that accepts the list of coefficients `L` and a real number `x_0` as arguments. It should return the polynomial evaluated at the value `x_0`. For example poly([1, 2, 3], 5) should return the value $1+2\times5+3\times5^2=86$.

```
defpoly(L,x_0):
'''
    Arguments:
        L: list of integers
        x_0: integer
    Return:
        result: integer
    '''
```

You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def poly(L, x_0):
    if len(L) == 1:
        return L[0]
    return L[0] + x_0 * poly(L[1: ], x_0)
```

PPA 13

Write a recursive function named `power` that accepts a square matrix A and a positive integer m as arguments and returns A^m.

```
defpower(A,m):
'''
    Arguments:
        A: list of lists
        m: integer
    Return:
        result: list of lists
```

```
'''
```

You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def zero_matrix(n):
    '''zero matrix of size n x n'''
    M = [ ]
    for i in range(n):
        row = [ ]
        for j in range(n):
row.append(0)
M.append(row)
    return M

def mat_mul(A, B):
    '''multiply A and B'''
    n = len(A)
    prod = zero_matrix(n)
    # multiply A and B
    for i in range(n):
        for j in range(n):
for k in range(n):
                prod[i][j] += A[i][k] * B[k][j]
    return prod

def power(A, m):
    '''Raise A to the power m'''
    if m == 1:
        return A
A_min_one = power(A, m - 1)
    return mat_mul(A_min_one, A)
```

PPA 14

Challenge Problem: If you have grasped the essence of recursion, then you would see the simplicity of the whole idea when you solve this problem. All those who have done Maths-2, or are doing Maths-2, should be able to appreciate this problem. If you are yet to do Maths-2, you can still try to attempt this problem. But we request learners not to be intimidated by the heavy use of notation. Feel free to skip this problem if you perceive it to be too hard.

Determinant
Any square matrix M*M* has a number associated with it called its determinant. The determinant of a 2×2 matrix is defined as follows:

$$det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

For any $n{\times}n$ square matrix M, its determinant is defined recursively as follows:

$$det(M) \ = \sum_{j=0}^{n-1} (-1)^j \cdot M[0][j] \cdot M_j$$

Here,M_j is the determinant of the matrix obtained by removing the 0^{th} row and the j^{th} column of M for $0 \le j < n$. We have used zero-based indexing.
For example, for a 3×3 matrix, we have:

$$det\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}\right)$$
$$= (-1)^0 \cdot a \cdot det\left(\begin{bmatrix} e & f \\ h & i \end{bmatrix}\right) + (-1)^1 \cdot b \cdot det\left(\begin{bmatrix} d & f \\ g & i \end{bmatrix}\right) + (-1)^2 \cdot c \cdot det\left(\begin{bmatrix} d & e \\ g & h \end{bmatrix}\right)$$

Write a recursive function named `det` that accepts a square matrix as argument and returns its determinant. In the process of writing this function, it would be useful to look into GrPA-3 of week-7. A good approach would be to write two functions: `det` and `minor_matrix`.

```
defdet(M):
'''
    Argument:
        M: list of lists
    Return:
        result: integer
  '''
```

You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def minor_matrix(M, col):
    n = len(M)
M_ij = [ ]
    for i in range(1, n):
        L = [ ]
        for j in range(n):
            if j == col:
                continue
L.append(M[i][j])
M_ij.append(L)

    return M_ij

def det(M):
    n = len(M)
    if n == 2:
        return M[0][0] * M[1][1] - M[0][1] * M[1][0]
dsum = 0
```

```
    for j in range(n):
dsum = dsum + M[0][j] * det(minor_matrix(M, j)) * ((-1) ** (j))
    return dsum
```

PPA 15

You have a locker that has a finite number of coins in it. Each coin has some positive integer that is engraved on it. This denotes how valuable the coin is. You wish to draw a subset of coins from the locker whose total worth is s*s*. Your task is to determine if this can be done with the coins available in your locker.

Write a recursive function named `subset_sum` that accepts a list of positive integers `L` and a positive integer `s` as arguments. The list `L` represents the coins in your locker. The integer `s` represents the total value of the coins that you need to withdraw. Return `True` if you can withdraw some subset of coins whose combined worth is s*s*, return `False` otherwise.

```
defsubset_sum(L,s):
'''
    Arguments:
        L: list of integers
        s: integer
    Return:
        result: bool
  '''

```

---

(1) If you need a hint, come to Discourse.

(2) You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def subset_sum(L, s):
    if s == 0:
        return True
    if len(L) == 0:
        return False
    if subset_sum(L[:-1], s - L[-1]):
        return True
    else:
        return subset_sum(L[:-1], s)
```

GrPA 1

Write a recursive function named `reverse` that accepts a list `L` as argument and returns the reversed list.

```
defreverse(L):
'''
    Arguments:
        L: list, type of elements could be anything
    Return:
        result: list
    '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def reverse(L):
    if len(L)==1:
        return L
    else:
        return ([L[-1]]+reverse(L[:-1]))
```

GrPA 2

Write a recursive function named `linear` that accepts the following arguments:

- `P`: a non-empty list of positive integers
- `Q`: a non-empty list of positive integers
- `k`: a positive integer

It should return `True` only if both the conditions given below are satisfied:

- P and Q are of same length.
- $P[i]=k \cdot Q[i]$, for every integer i in the range $[0,\text{len}(P)-1]$, endpoints inclusive.

Even if one of these conditions is not satisfied, it should return `False`.

```
deflinear(P,Q,k):
'''
    Arguments:
        P: list of integers
        Q: list of integers
        k: integer
    Return:
        result: bool
    '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def linear(P, Q, k):
    if len(P) != len(Q):
        return False
    if len(P) == 0:
        return True
    if P[0] / Q[0] != k:
        return False
    return linear(P[1: ], Q[1: ], k)
```

GrPA 3

The Collatz function is defined for a positive integer n as follows.

$$f(n) = \begin{cases} 3n+1 & if\ n\ is\ odd \\ n/2 & if\ n\ is\ even \end{cases}$$

We consider the repeated application of the Collatz function starting with a given integer nn, which results in the following sequence:

$$f(n), f(f(n)), f(f(f(n))), \ldots$$

It is conjectured that no matter which positive integer n you start from, the sequence will always reach 1. For example, Ifn=10, the sequence is:

Seq No.	n	$f(n)$
1	10	5
2	5	16
3	16	8
4	8	4
5	4	2
6	2	1

Thus, if you start from n=10, you need to apply the function f six times in order to first reach 1.

Write a recursive function named `collatz` that accepts a positive integer n as argument, where $1<n\leq 32{,}000$, and returns the number of times f has to be applied repeatedly in order to first reach 1.

```
defcollatz(n):
'''
    Argument:
        n: integer
        Assume that 1 < n <= 32,000
    Returns:
        result: integer
    '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def collatz(n):
    if n==2:
        return 1
    else:
        if n%2!=0:
            c=1+collatz((3*n)+1)
            return c
        else:
            c=1+collatz(n/2)
            return c
```

GrPA4

Fibonacci

Fibonacci is a young resident of the Italian city of Pisa. He spends a lot of time at the premises of the Leaning Tower of Pisa, one of the iconic buildings in the city, that is situated close to his home. During all his visits to the tower, he plays a strange game while climbing the marble steps of the tower.

The Game

Fibonacci likes to climb the steps either one at a time, two at a time or three at a time. This adds variety to the otherwise monotonous task of climbing. He wants to find the total number of ways in which he can climb *n* steps, assuming that the order of his individual steps matters. Your task is to help Fibonacci compute this number.

For example, if he wishes to climb three steps, the case of n = 3, he could do it in four different ways:

- (1, 1, 1): do it in three moves, one step at a time
- (1, 2): do it in two moves, first take a single step, then a double step
- (2, 1): do it in two moves, first take a double step, then a single step
- (3): do it in just one move, directly leaping to the third step

To take another example, if n = 5, then some of the sequences could be:

(1, 3, 1), (1, 1, 3), (3, 1, 1), (2, 1, 1, 1), (1, 2, 1, 1), (2,1,2),
(1,3,1),(1,1,3),(3,1,1),(2,1,1,1),(1,2,1,1),(2,1,2)

Each sequence is one of the ways of climbing five steps. The point to note here is that each element of a sequence can only be 1, 2 or 3.

Write a recursive function named `steps` that accepts a positive integer n as argument. It should return the total number of ways in which Fibonacci can ascend n steps. Note that the order of his steps is important.

```
defsteps(n):
'''
    Argument:
      n: integer
    Return:
      result: integer
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def steps(n):
    if n==1 or n==0:
        return 1
    if n==2:
        return 2
    else:
        return (steps(n-1)+steps(n-2)+steps(n-3))
```

GrPA 5

`P` is a dictionary of father-son relationships that has the following structure: for any `key` in the dictionary, its corresponding `value` is the father of `key`. As an example:

```
P={
'Jahangir':'Akbar',
'Akbar':'Humayun',
'Humayun':'Babur'
}
```

If `'Jahangir'` is the key, then the `'Akbar'`, his father, is the value. This is true of every key in the dictionary.

Write a recursive function named `ancestry` that accepts the following arguments:

- `P`: dictionary of relationships
- `present`: name of a person, string
- `past`: name of a person, string

It should return the sequence of ancestors of the person named `present`, traced all the way back up to person named `past`. For example, `ancestry(P, 'Jahangir', 'Babur')` should return the list:

L = ['Jahangir', 'Akbar', 'Humayun', 'Babur']

In more Pythonic terms, L[i] is the father of L[i - 1], for 1≤*i*<len(*L*), with the condition that L[0] should be `present` and L[-1] should be `past`.

```
defancestry(P,present,past):
'''
    Arguments:
        P: dict, key and value are strings
        present: string
        past: string
    Return:
        result: list of strings
  '''
```

---

(1) You can assume that no two persons in the dictionary have the same name. However, a given person could either appear as a `key` or as a `value` in the dictionary.
(2) A given person could appear multiple times as one of the values of the dictionary. For example, in test-case-2, Prasanna has two sons, Mohan and Krishna, and hence appears twice (as a value).

(2) You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def ancestry(P, present, past):
    if present==past:
        return [past]
    else:
        return ([present]+ancestry(P,P[present],past))
```

Week 9

PPA 1

Write a function named `read_file` that accepts a text file named `filename` as argument. Within the function, read the file and print each line of the file on a separate line in the console. You shouldn't print any extra characters at the end of a line. There shouldn't be an empty line between any two consecutive lines.

```
defread_file(filename):
'''
    Argument:
       filename: string, name of the file to be read
    Return:
       None
  '''
```

(1) `filename` is a string variable that holds the name of the file. For example, in the first test case, it is `filename = 'public_1.txt'`.
(2) You do not have to accept input from the console. You have to write the function definition and print the contents of the file within the function.

```
def read_file(filename):
    f = open(filename, 'r')
    for line in f:
        print(line.strip())
f.close()
```

PPA 2

Write a function named `read_line` that accepts a text file named `filename` and a positive integer *n* as arguments. Within the function, read the file and return the *n*th line of the file. If the file has fewer than *n* lines, return the string `'None'`.

```
defread_line(filename,n):
'''
    Argument:
       filename: string, name of the file to be read
    Return:
       string: return nth line of the file
  '''
```

(1) `filename` is a string variable that holds the name of the file. For example, in the first test case, it is `filename = 'public_1.txt'`.

(2) You do not have to accept input from the console or print the output. You have to write the function definition.

```
def read_line(filename, n):
    f = open(filename, 'r')
    count = 0
    line = f.readline()
    while line != '':
        count += 1
        line = line.strip()
        if count == n:
            return line
        line = f.readline()
f.close()
    return None
```

PPA 3

Write a function named `get_max_line` that accepts a text file named `filename` as argument. Each line in this file contains an integer. The function should return the line number that houses the maximum integer in the file. If multiple lines have the same maximum number, return the smaller of the two. Line numbers start from one and not zero.

```
defget_max_line(filename):
'''
    Argument:
      filename: string, path to the file
    Return:
      result: integer
  '''
```

(1) `filename` is a string variable that holds the name of the file. For example, in the first test case, it is `filename = 'public_1.txt'`.
(2) You do not have to accept input from the console or print the output. You have to write the function definition.

```
def get_max_line(filename):
    f = open(filename, 'r')
    line = f.readline()
    maxnum, maxline = int(line.strip()), 1
    count = 1
    for line in f:
        num = int(line.strip())
        count += 1
        if num > maxnum:
            maxnum = num
            maxline = count
    f.close()
    return maxline
```

PPA 4

`filename` **points to a CSV file that has two columns. The first column is the name of a student, the second column is this student's age. The first line of the file will always be the header. A sample file is given below for your reference:**

```
Name,Age
Arti,20
Kalam,60
Atul,25
Krishnan,24
Sahana,20
```

Write a function named `get_dict` **that accepts the** `filename` **as argument and returns a dictionary where the keys are the student names and the values are the corresponding ages of the students.**

```
defget_dict(filename):
'''
    Argument:
      filename: string, path of the file
    Return:
      result: dict; keys are strings, values are integers
  '''
```

---

(1) `filename` **is a string variable that holds the name of the file. For example, in the first test case, it is** `filename = 'public_1.txt'`**.**
(2) You do not have to accept input from the console or print the output. You have to write the function definition.

```
def get_dict(filename):
    f = open(filename, 'r')
    f.readline()
    P = dict()
    for line in f:
        name, age = line.strip().split(',')
        age = int(age)
        P[name] = age
    f.close()
    return P
```

PPA5

`filename` **points to a file that contains a n \times n$n \times n$ matrix. The i^{th}ith line of the file represents the (i - 1)^{th}$(i-1)th$ row of the matrix as a sequence of comma separated integers, where 1 \leq i \leq n$1 \leq i \leq n$. We have used zero-based indexing here. This file doesn't have a header and has exactly nn lines. A sample file is given for your reference:**

```
1,2,3
4,5,6
7,8,9
```

Your task is to return the matrix [[1, 2, 3], [4, 5, 6], [7, 8, 9]].

Write a function named `get_matrix` that accepts the `filename` as argument. It should return the matrix as a list of lists. Each cell of the matrix should be an integer and not a string.

```
defget_matrix(filename):
'''
    Argument:
      filename: string
    Return:
      matrix: list of lists
  '''
```

(1) `filename` is a string variable that holds the name of the file. For example, in the first test case, it is `filename = 'public_1.txt'`.
(2) You do not have to accept input from the console or print the output. You have to write the function definition.

```
def get_matrix(filename):
    f = open(filename, 'r')
    M = [ ]
    for line in f:
        row = line.split(',')
        for i in range(len(row)):
            row[i] = int(row[i])
        M.append(row)
    f.close()
    return M
```

PPA 6

A sequence of nn terms (a_1, a_2, \dots, a_n)($a1,a2,…,an$) is called an Arithmetic progression (AP) if the difference between any two consecutive terms stays constant. That is:

a_2 - a_1 = a_3 - a_2 = \cdots = a_n - a_{n - 1} = d$a2−a1=a3−a2=⋯=an−an−1=d$

Some useful terms:

- a_1$a1$ is the first term of the AP.
- dd is called the common difference of the AP.
- nn is the total number of terms in the AP.

Write a function named `write_AP` that accepts the following arguments:

- `a_1`: first term of the AP, integer.
- `d`: common difference of the AP, integer.
- `n`: number of terms in the AP, positive integer.

Within the function, create a file named `out.txt` and write the first nn terms of the AP to it, one term on each line, starting from the first term.

```
defwrite_AP(a_1,d,n):
```

```
'''
    Arguments:
      a_1: first term, integer
      d: common difference, integer
      n: number of terms, integer
    Return:
      None
  '''
```

---

(1) The last line of the file should not end with a '\n'. The last character of every other line in the file should end with a '\n'. This is a convention that we will be following in all questions that ask you to write to a file.
(2) You do not have to accept input from the console or print the output. You have to write the function definition. Within the function, create the file named `out.txt`.

```python
def write_AP(a_1, d, n):
    f = open('out.txt', 'w')
    x = a_1
    for i in range(n):
        line = f'{x}'
        if i != n - 1:
            line = line + '\n'
        f.write(line)
        x = x + d
    f.close()
```

PPA 7

`filename` points to the name of some text file. Each line in this file is missing a period at the end of the line. Write a function named `add_period` that accepts `filename` as argument and creates a new file named `out.txt`. The function should copy the contents of `filename` into `out.txt` and add a period at the end of each line.

```
defadd_period(filename):
'''
    Argument:
      filename: string; path of the file
    Return:
      None
  '''
```

---

(1) `filename` is a string variable that holds the filename. For example, in the first test case, it is `public_1.txt`.

(2) The last line of the file should not end with a '\n'. The last character of every other line in the file should end with a '\n'. This is a convention that we will be following in all questions that ask you to write to a file.
(3) You do not have to accept input from the console. You have to write the function definition and within the function, create a file named `out.txt` according to the required specification.

```python
def add_period(filename):
    f = open(filename, 'r')
    g = open('out.txt', 'w')
    for line in f:
        line = line.strip()
        g.write(line + '.' + '\n')
    f.close()
    g.close()
```

PPA 8

The scores of a class of students in the online degree program is represented as a CSV file with the following header:

```
Name,Gender,CT,Python,PDSA
```

The name of the file is given by the variable `filename`. The first line will be the header.

Write a function named `improvement` which accepts the `filename` as argument. It should return the number of students whose scores have increased across the three courses. That is, the number of students whose scores are in this order: CT < Python < PDSA.

```python
defimprovement(filename):
'''
    Argument:
       filename: string, path to file
    Return:
       count: integer
  '''
```

---

(1) `filename` is a string variable that holds the filename. For example, in the first test case, it is `public_1.txt`.
(2) You do not have to accept input from the console. You just have to write the function definition.

```python
def improvement(filename):
    f = open(filename, 'r')
    f.readline()
    count = 0
    for line in f:
        name, gender, ct, python, pdsa = line.strip().split(',')
        ct, python, pdsa = int(ct), int(python), int(pdsa)
        if ct < python < pdsa:
            count += 1
    f.close()
    return count
```

PPA 9

The scores of a class of students in the online degree program is represented as a CSV file with the following header:

```
SeqNo,Name,Gender,CT,Python,PDSA
```

The name of the file is given by the variable `filename`. The first line will be the header. The contents of the file will be in increasing order of sequence numbers.

Write a function named `extract_lines` that accepts `filename` as argument. Within the function, read the file and look for all male students who have scored at least 70 marks in Python. Copy these lines into a new file named `python.csv`. The entries in this file should be in the increasing order of sequence numbers. Also, the first line of `python.csv` should be the header, which is same as the one in `filename`.

```
defextract_lines(filename):
'''
    Argument:
      filename: string
    Return:
      None
  '''
```

---

(1) `filename` is a string variable that holds the filename. For example, in the first test case, it is `public_1.csv`.
(2) You do not have to accept input from the console. You just have to write the function definition.

(3) We shall be printing a random selection of five entries from `python.csv`.

(4) The last line of the file you write to should not end with a '\n'. The last character of every other line in the file should end with a '\n'. This is a convention that we will be following in all questions that ask you to write to a file.

```python
def extract_lines(filename):
    f = open(filename, 'r')
    header = f.readline()
    g = open('python.csv', 'w')
    g.write(header)
    for line in f:
        L = line.strip().split(',')
        python = int(L[4])
        gender = L[2]
        if gender == 'M' and python >= 70:
            g.write(line)
    f.close()
    g.close()
```

PPA 10

Write a function named `number_grid` that accepts two positive integers mm and nn as arguments. Within the function, create a file named `numgrid.csv`. Write the first mnmn positive integers to the file in the following way:

- Each line should be a sequence of nn comma-separated integers.
- There should be a total of mm lines in the file.

For example, for the case of m = 5, n = 3m=5,n=3, the file should be:

```
1,2,3
4,5,6
7,8,9
10,11,12
13,14,15
```

```
defnumber_grid(m,n):
'''
    Arguments:
        m, n: positive integers
    Return:
        None
  '''
```

---

(1) The last line of the file should not end with a '\n'. The last character of every other line in the file should end with a '\n'. This is a convention that we will be following in all questions that ask you to write to a file.

(2) You do not have to accept input from the console. You just have to write the function definition.

```
def number_grid(m, n):
    f = open('numgrid.csv', 'w')
    x = 1
    line = ''
    while x <= m * n:
        line = line + str(x)
        if x % n == 0:
            if x != m * n:
                line = line + '\n'
            f.write(line)
            line = ''
        else:
            line = line + ','
        x = x + 1
    f.close()
```

PPA 11

The scores dataset is a list of dictionaries one of whose entries is given below for your reference:

```
{'SeqNo':0,'Name':'Devika','Gender':'F','City':'Bengaluru',
'Mathematics':85,'Physics':100,'Chemistry':79}
```

Write a function named `collection_to_file` that accepts the `scores_dataset` as argument. Within the function, create a CSV file named `scores.csv` with the following header:
SeqNo,Name,Gender,City,Mathematics,Physics,Chemistry
Each line in the file after the header should have the details of one student. The file should be arranged in increasing order of `SeqNo`. To simplify things, `scores_dataset` is already sorted in the increasing order of `SeqNo`.

```
defcollection_to_file(scores_dataset):
'''
    Argument:
      scores_dataset: list of dicts
    Return:
      None
  '''
```

---

(1) The last line of the file should not end with a '\n'. The last character of every other line in the file should end with a '\n'. This is a convention that we will be following in all questions that ask you to write to a file.

(2) You do not have to accept input from the console. You just have to write the function definition.

(3) We shall be printing a random selection of five entries from `scores.csv`.

```
def collection_to_file(scores_dataset):
    f = open('scores.csv', 'w')
    f.write('SeqNo,Name,Gender,City,Mathematics,Physics,Chemistry\n')
    for stud in scores_dataset:
        sno = stud['SeqNo']
        name = stud['Name']
        gender = stud['Gender']
        city = stud['City']
        ma = stud['Mathematics']
        phy = stud['Physics']
        chm = stud['Chemistry']
        line = f'{sno},{name},{gender},{city},{ma},{phy},{chm}\n'
        ifstud['SeqNo'] == scores_dataset[-1]['SeqNo']:
            line = line.strip()
        f.write(line)
    f.close()
```

PPA 12

Write a function named `write_pattern` that accepts an odd positive integer nn as argument. Within the function, create a file named `pattern.csv` and write an n \times nn$\times n$ matrix to it, such that it has ones on both the diagonals and zeros in all

other places. Each row of the matrix should be written as a sequence of comma-separated numbers.

```
defwrite_pattern(n):
'''
    Argument:
        n: integer
    Return:
        None
  '''
```

---

(1) The last line of the file should not end with a '\n'. The last character of every other line in the file should end with a '\n'. This is a convention that we will be following in all questions that ask you to write to a file.

(2) You do not have to accept input from the console. You just have to write the function definition.

```
def write_pattern(n):
    f = open('pattern.csv', 'w')
    for i in range(n):
        for j in range(n):
            if j == i or j == n - i - 1:
                f.write('1')
            else:
                f.write('0')
            if j != n - 1:
                f.write(',')
        if i != n - 1:
            f.write('\n')
    f.close()
```

GRPA 1

`filename` is a text file that contains a collection of words in lower case, one word on each line. Write a function named `get_freq` that accepts `filename` as argument. It should return a dictionary where the keys are distinct words in the file, the values are the frequencies of these words in the file.
For example, given the following file:

```
good
great
good
work
work
```

The dictionary returned should be:

```
{'good':2,'great':1,'work':2}
```

The order in which the words are added to the dictionary doesn't matter.

```
defget_freq(filename):
'''
    Argument:
      filename: string, path to file
    Return:
      result: dictionary; keys are strings, values are integers
  '''
```

(1) `filename` is a string variable that holds the name of the file. For example, in the first test case, it is `filename = 'public_1.txt'`.
(2) You do not have to accept input from the console or print the output to the console. You just have to write the function definition.

```
def get_freq(filename):
    f = open(filename, 'r')
    freq = dict()
    for line in f:
        word = line.strip()
        if word not in freq:
            freq[word] = 0
        freq[word] += 1
    f.close()
    return freq
```

GRPA 2

You are given two non-empty text files `file1` and `file2` that have f_1*f*1 and f_2*f*2 lines respectively. Each file is a collection of ten-digit phone numbers, one number per line. It is also known that 0 < f_1 \leq f_20<*f*1≤*f*2.
The following relations are defined on these two files:

- Subset: `file1` is a subset of `file2` if the phone number in the i^{th}*ith* line of `file1` is equal to the number in the i^{th}*ith* line of `file2`, for 1 \leq i \leq f_11≤*i*≤*f*1, and f_1 < f_2*f*1<*f*2.
- Equal: `file1` is equal to `file2` if the phone number in the i^{th}*ith* line of `file1` is equal to the number in the i^{th}*ith* line of `file2` for 1 \leq i \leq f_11≤*i*≤*f*1, and f_1 = f_2*f*1=*f*2.

Write a function named `relation` that accepts these two text files as arguments. It should return the string `Subset` if `file1` is a subset of `file2`. It should return `Equal` if `file1` is equal to `file2`. If both these conditions are not satisfied, it should return the string `No Relation`.

```
defrelation(file1,file2):
'''
    Arguments:
      file1, file2: strings, paths to two files
    Return:
```

```
    result: string --- 'Equal', 'Subset' or 'No Relation'
  '''
```

---

(1) `file1` and `file2` are string variables that hold the names of the two files. For example, in the first test case, it is `file1 = 'public_1_1.txt'` and `file2 = 'public_1_2.txt'`.
(2) You do not have to accept input from the console or print the output to the console. You just have to write the function definition.

(3) The strings `'9876543210'` and '9876543210\n' are not the same, though the phone numbers are the same. So, strip the strings of all new lines using the `strip` method before checking for equality of two lines across the files.

```
def relation(file1,file2):
    f1 = open(file1,'r')
    f2 = open(file2,'r')
    l1= f1.readlines()
    l2 =f2.readlines()
    f1.close()
    f2.close()
    flag = True
    if l1==l2:
        return'Equal'
    else:
        for i in range(len(l1)):
            ifl1[i] not in l2[i]:
                flag = False
    if flag:
        return'Subset'
    else:
        return'No Relation'
```

GRPA 3

`filename` is a CSV file that has the following header:

```
Player,Country,Goals
```

The first five lines of a sample file are given below:

```
Player,Country,Goals
P1,Brazil,20
P2,Argentina,30
P3,Brazil,50
P4,Germany,30
```

Write a function named `get_goals` that accepts `filename` and the name of a `country` as arguments. It should return a tuple having two elements: `(num_players, num_goals)`. `num_players` is the number of players from this country that appear in this file, `num_goals` is the total number of goals scored by all the players who belong to this country. If the country is not present in the file, then return the tuple `(-1, -1)`. In the example given above, Brazil is represented by 2 players who have scored a total of 70 goals for their country.

```
defget_goals(filename,country):
```

```
'''
    Arguments:
      filename: string
      country: string
    Return:
      result: tuple, (integer, integer)
  '''
```

---

(1) `filename` is a string variable that holds the name of the file. For example, in the first test case, it is `filename = 'public_1.csv'`.
(2) Each player who represents a country has scored at least one goal. That is, the last column in the file will have only positive integers.

(3) You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def get_goals(filename, country):
    f=open(filename,'r')
    l=f.readlines()[1:]
    c=-1
    g=-1
    for i in l:
        if country in i:
            t = i.split(',')
            if c == -1:
                c=1
                g = int(t[2])
            else:
                c+=1
                g+=int(t[2])
    f.close()
    return((c,g))
```

GRPA 4

Write a function named `num_to_words` that accepts a square matrix of single digit numbers as argument. Within the function, create a file named `words.csv`. Write the matrix to the file by replacing the digits with their corresponding words. For example, num_to_words([[1, 2], [3, 4]]) should create the file `words.csv` with the following contents:

```
one,two
three,four
```

Note that the matrix will only have integers from 00 to 99, endpoints inclusive.

```
defnum_to_words(mat):
'''
    Argument:
      mat: list of lists
    Return:
```

```
    None
'''
```

---

(1) You do not have to accept input from the user or print the output to the console. You just have to write the function definition. However, within the function, you have to create a file named `words.csv` and write to it.
(2) The last line of the file should not end with a '\n'. The last character of every other line in the file should end with a '\n'. This is a convention that we will be following in all questions that ask you to write to a file.

```
def num_to_words(mat):
    P = {0: 'zero', 1: 'one', 2: 'two',
    3: 'three', 4: 'four', 5: 'five',
    6: 'six', 7: 'seven', 8: 'eight',
    9: 'nine'}
    f = open('words.csv', 'w')
    n = len(mat)
    for i in range(n):
        for j in range(n):
            line = f'{P[mat[i][j]]}'
            if j != n - 1:
                line += ','
            f.write(line)
        if i != n - 1:
            f.write('\n')
    f.close()
```

Week 10

PPA1

Write a class named `Student` with the following specification:
Attributes

- `name`: string, denotes the name of the student
- `marks`: int, denotes the marks obtained by the student in some exam

Methods
`self` is the first argument of all methods. We will only mention the additional arguments, if any.

- `__init__`: constructor with two arguments — `name` and `marks`; assign these two values to the corresponding attributes within the constructor
- `print_info`: prints the name and the marks of the student separated by a colon.

You do not have to accept input from the user or print output to the console. You just have to define the class based on the specification given in the question.

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks
    def print_info(self):
        print(f'{self.name}:{self.marks}')
```

PPA 2

Consider a class named `Word` that is given to you as a part of the prefix code. Your task is to create an object of the class by accepting input from the console.

The first line of input is a word. The second line of input is its part of speech.

(1) Create an object of the class `Word` and name it `word` using the values that you have accepted as input.
(2) Call the method `print_info` using this object.

---

(1) You have to accept input from the console. Calling the method `print_info` will take care of printing the required details.
(2) There will be three lines of input. Ignore the third one. This is to help us in evaluating your output.

```
class Word:
    count = 0
    def __init__(self, word, pos):
        Word.count += 1
        self.word = word
        self.pos = pos

    def print_info(self):
        print(f'The word is "{self.word}" and its part of speech is "{self.pos}".')

inp_1 = input()
inp_2 = input()
word = Word(inp_1, inp_2)
word.print_info()
```

PPA 3

We are going to model points in 2D space as objects of a class named `Point`. Mathematical details that are relevant to solve this problem are given below:

- Distance of a point P(x, y) from the origin is $\sqrt{x^2 + y^2}$.
- Slope of the line joining the origin and P is y/x if x≠0.

Define a class named `Point` that has the following specification:
<u>Attributes</u>

- `x`: int, x-coordinate of the point in 2D space.
- `y`: int, y-coordinate of the point in 2D space.

<u>Methods</u>

`self` is the first argument of all methods. We will only mention the additional arguments, if any.

- `__init__`: constructor with two arguments — `x` and `y`; assign these two values to the corresponding attributes within the constructor
- `distance`: return the distance of the point from the origin as a float value
- `is_origin`: return `True` if the point coincides with the origin, and `False` otherwise
- `on_xaxis`: return `True` if the point is on the x-axis, and `False` otherwise
- `on_yaxis`: return `True` if the point is on the y-axis and `False` otherwise
- `quadrant`: return the quadrant that this point belongs to; assume that this method will only be called if the point is not on either of the axes; possible return values are ['first', 'second', 'third', 'fourth'].
- `slope`: return the slope of the line joining the origin and this point as a float value; assume that this method will only be called if the point is not on the y-axis

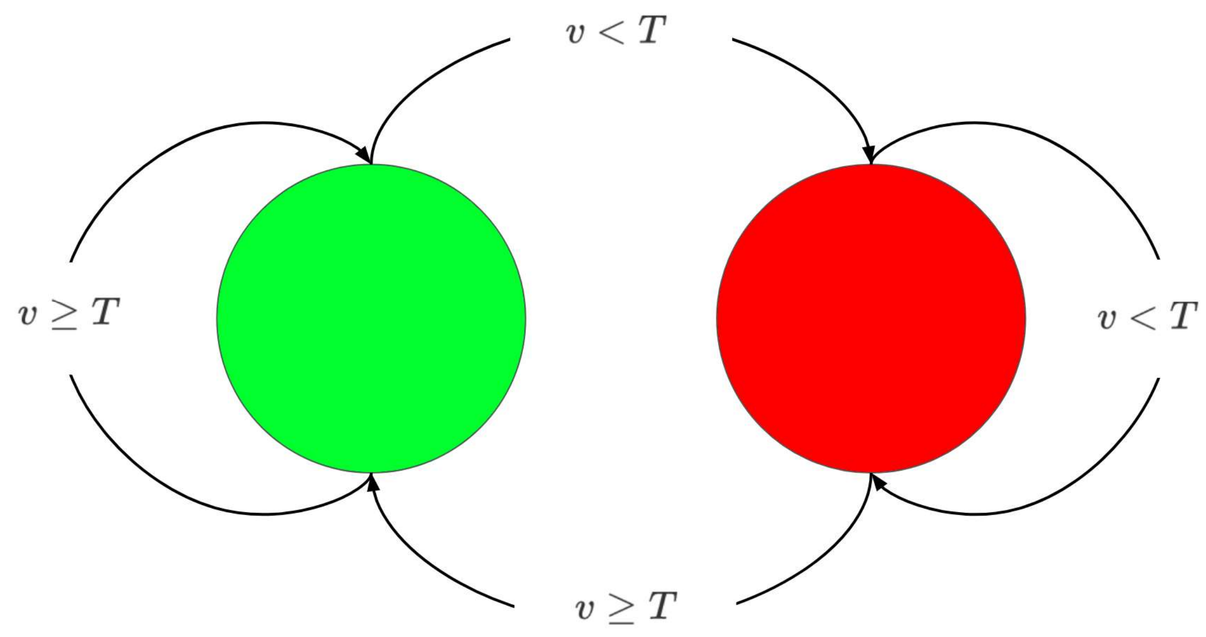
---

(1) Each test case corresponds to one method call. We will use `P` to denote the name of the object.
(2) You do not have to accept input from the user or print output to the console. You just have to define the class based on the specifications given in the question.

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance(self):
        return pow(self.x ** 2 + self.y ** 2, 0.5)

    def is_origin(self):
        return self.x == 0 and self.y == 0

    def on_xaxis(self):
        return self.y == 0

    def on_yaxis(self):
        return self.x == 0

    def quadrant(self):
        if self.x > 0 and self.y > 0:
            return 'first'
        if self.x < 0 and self.y > 0:
            return 'second'
        if self.x < 0 and self.y < 0:
            return 'third'
        if self.x > 0 and self.y < 0:
            return 'fourth'

    def slope(self):
        return self.y / self.x
```

PPA 4

Consider an intelligent traffic signal. The signal has two states: red or green. The vehicle density in front of the signal is denoted by the variable v. If the vehicle density crosses a threshold T in either direction, the state of the signal changes. The dynamics of this change is represented in the image given below:



For example, if the signal is currently red, and the vehicle density becomes greater than or equal to the threshold, it is time to turn the signal green. This is denoted by the arrow from red to green at the bottom of the image. Assume that the signal *senses* the vehicle density every 30 seconds and *updates* its state appropriately.

Write a class named `Signal` with the following specification:

<u>Attributes</u>

- **`state`: string, either "red" or "green"; represents the current state of the signal**
- **`v`: int, represents the vehicle density at the current instant (v is in lower case)**
- `T`: int, threshold for the vehicle density (T is in upper case)

<u>Methods</u>

`self` is the first argument of all methods. We will only mention additional arguments, if any.

- `__init__`: constructor; accepts the threshold `T` as argument; initially the signal is red and the vehicle density is 0.
- `sense`: accept the vehicle density as argument and update the corresponding attribute; assume that this information comes from a sensor.
- `update`: update the state of the signal-attribute depending on the current values of the attributes.

(1) Each test case corresponds to one or more method calls. We will use `S` to denote the name of the object.
(2) You do not have to accept input from the user or print output to the console. You just have to define the class based on the specification given in the question.

```
class Signal:
    def __init__(self, T):
        self.state = 'red'
        self.v = 0
        self.T = T

    def sense(self, v):
        self.v = v

    def update(self):
        if self.v >= self.T:
            if self.state == 'red':
                self.state = 'green'
        else:
            if self.state == 'green':
                self.state = 'red'
```

PPA 5

A vector in 2D space corresponding to the point P(x, y) has one end at the origin and the other end at the point P. We are interested in the following operations on a vector:
Scaling
A vector can be scaled by a number s. For example, if [x, y] is a vector, then scaling it by s transforms it into the vector [s . x, s . y].
Reflection
If a vector [x, y] is reflected about the Y-axis, it becomes the vector [-x, y]. Likewise, if it is reflected about the X-axis, it becomes the vector [x, -y].
Addition
Two vectors [a, b] and [c, d] can be added to give the vector [a + c, b + d].

Write a class named `Vector` that has the following specification:
Attributes
- `x`: int, first coordinate of the vector
- `y`: int, second coordinate of the vector

Methods
- `__init__`: constructor with two arguments — `x` and `y`; assign these two values to the attributes within the constructor
- `print_info`: print a string that represents the coordinates of the current vector in the following form: `(x,y)`; there should be no spaces anywhere in the string
- `scale`: accept an integer `s` as argument and scale the current vector by s units

- `reflect_about_X`: reflect the current vector about the X-axis
- `reflect_about_Y`: reflect the current vector about the Y-axis
- `add`: accept another `Vector` as argument, and return the sum of the current vector (`self`) and this argument; you must return an object of type `Vector`

---

(1) Each test case corresponds to one or more method calls. We will use `V` to denote the name of the object.
(2) You do not have to accept input from the user or print output to the console. You just have to define the class based on the specifications given in the question.

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def reflect_about_X(self):
        self.y *= -1

    def reflect_about_Y(self):
        self.x *= -1

    def scale(self, s):
        self.x, self.y = s * self.x, s * self.y

    def add(self, V):
        v = Vector(0, 0)
        v.x = self.x + V.x
        v.y = self.y + V.y
        return v

    def print_info(self):
        return f'({self.x},{self.y})'
```

PPA 6

Consider a class named `Country` that is defined in the prefix code. Your task is to create a list of objects of type `Country`.

The first line of input is a positive integer n that denotes the number of countries. n blocks of input follow. Each block of input will have two lines; the first line will be the name of the country and the second line will be its capital. Corresponding to each block, create an object of type `Country`. Append each object to a list named `countries`. That is, each element of the list should be an object of type `Country`.

You have to process 2n + 1 lines of input in all. In addition, we will have one final line of input. Ignore this. This will help us in evaluating your output.

```
n = int(input())
countries = [ ]
for _ in range(n):
    name = input()
    capital = input()
    cnt = Country()
    cnt.set_name(name)
    cnt.set_capital(capital)
    countries.append(cnt)
```

PPA 7

`Question` is a class that is defined in the prefix code. Define a class named `NAT` that is a sub-class of `Question` with the following specification:

Attributes

Only those attributes that are specific to the derived class are mentioned below. The rest have to be inherited from the base class.

- `answer`: int, numerical answer for this question

Methods

Only those methods that are specific to the derived class are mentioned below. The rest have to be inherited from the base class.

- `__init__`: Accept the arguments `statement`, `marks` and `answer`. Call the constructor of the base class with the first two arguments, and assign `answer` to the corresponding attribute of the derived class.
- `update_answer`: Accept an argument named `answer` and update the corresponding attribute of the class with this new answer.

(1) Each test case corresponds to one or more method calls. We will use `Q` to denote the name of the object.
(2) You do not have to accept input from the user or print output to the console. You just have to define the class based on the specification given in the question.

```
class NAT(Question):
    def __init__(self, statement, marks, answer):
        super().__init__(statement, marks)
        self.answer = answer

    def update_answer(self, answer):
        self.answer = answer
```

PPA 8

`Question` is a class that is defined in the prefix code. Define a class named `MCQ` that is a sub-class of `Question` with the following specification:

<u>Attributes</u>

Only those attributes that are specific to the derived class are mentioned below. The rest have to be inherited from the base class.

- `ops`: list of strings; list of options and will always have four elements
- `c_ops`: list of strings, list of correct options, it will be a subset of ['a', 'b', 'c', 'd'].

<u>Methods</u>

Only those methods that are specific to the derived class are mentioned below. The rest have to be inherited from the base class.

- `__init__`: Accept the arguments `statement`, `marks` and `ops` and `c_ops`. Call the constructor of the base class with the first two arguments, assign `ops` and `c_ops` to the corresponding attributes of the derived class.
- `print_question`: Since this method is already present in the base class, override it in the following way: print the statement of the question on the first line, followed by the four options on separate lines.

---

(1) Each test case corresponds to one or more method calls. We will use `Q` to denote the name of the object.
(2) You do not have to accept input from the user or print output to the console. You just have to define the class based on the specifications given in the question.

```
class MCQ(Question):
    def __init__(self, statement, marks, ops, c_ops):
        super().__init__(statement, marks)
        self.ops = ops      # list of all options
        self.c_ops = c_ops  # list of correct options

    def print_question(self):
        super().print_question()
        for i in range(4):
            print(self.ops[i])
```

PPA 9

For an introduction to plotting with Matplotlib, refer to this [Google Colab notebook](). You don't have to submit any code in the portal for this problem. You can create a local copy of this file and start practicing these problems.

Plot the following functions using Matplotlib, each on a separate graph.

- f(x) = 3x - 4$f(x)=3x-4$
- f(x) = x^2 + 2x - 15$f(x)=x2+2x-15$
- f(x) = 5(x - 1)(x - 2)(x - 3)$f(x)=5(x-1)(x-2)(x-3)$

- f(x) = e^xf(x)=ex
- f(x) = \log xf(x)=logx
- f(x) = \sin xf(x)=sinx

PPA 10

For an introduction to plotting with Matplotlib, refer to this Google Colab notebook. You don't have to submit any code in the portal for this problem. You can create a local copy of this file and start practicing these problems.

Plot the functions f(x) = \sin xf(x)=sinx and g(x) = \cos xg(x)=cosx on the same graph.

PPA 11

For an introduction to plotting with Matplotlib, refer to this Google Colab notebook. You don't have to submit any code in the portal for this problem. You can create a local copy of this file and start practicing these problems.

Consider a file named `scores.csv` that contains the scores of 25 students in Mathematics and Physics.

```
Maths,Physics
20,80
10,30
40,30
20,30
10,5
80,90
99,100
76,84
29,100
100,30
95,92
100,100
70,74
65,88
90,93
89,91
20,40
10,30
20,25
15,34
35,25
50,70
45,55
34,43
60,67
```

Construct a scatter plot with Mathematics scores on the X-Axis and Physics scores on the Y-Axis. You can use `plt.scatter` for this purpose. Do you observe anything?

PPA 12

For an introduction to plotting with Matplotlib, refer to this [Google Colab notebook](). You don't have to submit any code in the portal for this problem. You can create a local copy of this file and start practicing these problems.

Perform the following experiment:

(1) Throw a standard, unbiased, six-sided dice n*n* times.
(2) Get the frequency of occurrence of the numbers from 11 to 66.
(3) Convert the frequencies to probabilities.

(4) Represent the probabilities of obtaining each of the six numbers in the form of a bar plot. You can use `plt.bar` for this purpose.
Run this experiment for the following values of n*n*: 10, 100, 1000, 10000, 100000, 100000010,100,1000,10000,100000,1000000 and note down your observations.

PPA 13

For an introduction to plotting with Matplotlib, refer to this [Google Colab notebook](). You don't have to submit any code in the portal for this problem. You can create a local copy of this file and start practicing these problems.

The equation of a circle centered at the origin of radius r*r* is given by the following equation:
x^2 + y^2 = r^2$x2+y2=r2$
Draw a circle of radius 5 using Matplotlib using a scatter plot.

GRPA 1

Create a class named `Calculator` that has the following specification:
<u>Attributes</u>
`a`: int, we shall call this the first attribute
`b`: int, we shall call this the second attribute
<u>Methods</u>

- **`__init__`: accept two arguments `a` and `b`, assign them to the corresponding attributes**
- **`add`: return the sum of the two attributes**
- `multiply`: return the product of the two attributes
- `subtract`: subtract the second attribute from the first and return this value
- `quotient`: return the quotient when the first attribute is divided by the second attribute
- `remainder`: return the remainder when the first attribute is divided by the second

(1) Each test case corresponds to one or more method calls. We will use `C` to denote the name of the object.
(2) You do not have to accept input from the user or print output to the console. You just have to define the class based on the specifications given in the question.

```
class Calculator:
    def __init__(self, a, b):
        self.a, self.b = a, b
    def add(self):
        return self.a + self.b
    def multiply(self):
        return self.a * self.b
    def subtract(self):
        return self.a - self.b
    def quotient(self):
        return self.a // self.b
    def remainder(self):
        return self.a % self.b
```

GRPA 2

Create a class named `StringManipulation` that has the following specification:
Attributes
`words`: list of strings, all of which will be in lower case
Methods

- **`__init__`: accept a list `words` as argument and assign it to the corresponding attribute**
- **`total_words`: return the total number of words in `words`**
- `count`: accept an argument named `some_word` and return the number of times this word occurs in the list `words`
- **`words_of_length`: accept a positive integer `length` as argument and return a list of all the words in the list `words` that have a length equal to `length`**
- **`words_start_with`: accept a character `char` as argument and return the list of all the words in `words` that start with `char`**
- `longest_word`: return the longest word in the list `words`; if there are multiple words that satisfy this condition, return the first such occurrence
- `palindromes`: return a list of all the words that are palindromes in `words`

(1) For those methods where you are expected to return a list of words, make sure that the words in the returned list appear in the order in which they are present in the attribute `words`.
(2) Each test case corresponds to one or more method calls. We will use `S` to denote the name of the object.
(3) You do not have to accept input from the user or print output to the console. You just have to define the class based on the specifications given in the question.

```
class StringManipulation:
    def __init__(self, words):
        self.words = words
    def total_words(self):
        return len(self.words)
    def count(self, some_word):
        n = 0
        for word in self.words:
            if word == some_word:
                n += 1
        return n
    def words_of_length(self, length):
        L = [ ]
        for word in self.words:
            if len(word) == length:
                L.append(word)
        return L
    def words_start_with(self, char):
        L = [ ]
        for word in self.words:
            if word[0] == char:
                L.append(word)
        return L
    def longest_word(self):
        maxword = self.words[0]
        for word in self.words:
            if len(word) > len(maxword):
                maxword = word
        return maxword
    def palindromes(self):
        L = [ ]
        for word in self.words:
            reverse = ''
            for char in word:
                reverse = char + reverse
            if word == reverse:
                L.append(word)
        return L
```

GRPA 3

A class named `Shape` is given to you as a part of the prefix code. Write a class named `Square` that is derived from `Shape` with the following specification:
Attributes
Only those attributes that are specific to the derived class are mentioned below. The rest have to be inherited from the base class.

- `side`: int, side of the square
  Methods
  Only those methods that are specific to the derived class are mentioned below. The rest have to be inherited from the base class.

- `__init__`: accept `side` as an argument:
- Call the constructor of the base class and set the `name` attribute to "Square" using it.
- Assign `side` to the corresponding attribute of this class.

- Call the methods `compute_area` and `compute_perimeter` within the constructor.
- `compute_area`: compute the area of the square and assign it to the attribute `area`.
- `compute_perimeter`: compute the perimeter of the square and assign it to the attribute `perimeter`.

---

(1) Each test case corresponds to one or more method calls. We will use `S` to denote the name of the object.
(2) You do not have to accept input from the user or print output to the console. You just have to define the class based on the specifications given in the question.

```
class Shape:
    def __init__(self, name):
        self.name = name
        self.area = None
        self.perimeter = None

    def display(self):
        print(f'{self.name} has an area of {self.area} and perimeter of
{self.perimeter}')
class Square(Shape):
    def __init__(self, side):
        super().__init__('Square')
        self.side = side
        self.compute_area()
        self.compute_perimeter()

    def compute_area(self):
        self.area = self.side ** 2

    def compute_perimeter(self):
        self.perimeter = 4 * self.side
```

GRPA 4

Create a class `Time` with the following specification:
<u>Attributes</u>

- `time`: int, represents time in seconds

<u>Methods</u>

- **`__init__`: accept `time` in seconds as an argument and assign it to the corresponding attribute**
- **`seconds_to_minutes`: convert the value of `time` into minutes and return a string in the format: "<minutes> min <seconds> sec". For example: if the value of the attribute `time` is 170170, this method should return the string "2 min 50 sec"**
- `seconds_to_hours`: convert the value of `time` into hours and return a string in the format: "<hours> hrs <minutes> min <seconds> sec". For example: if the value of the attribute `time` is 1089010890, this method should return the string "3 hrs 1 min 30 sec"
- `seconds_to_days`: convert the value of `time` into days and return a string in the format: "<days> days <hours> hrs <minutes> min <seconds> sec". For example: if the value of the

attribute `time` is 8646086460, this method should return the string "1 days 0 hrs 1 min 0 sec"

---

(1) Each test case corresponds to one or more method calls. We will use `S` to denote the name of the object.
(2) You do not have to accept input from the user or print output to the console. You just have to define the class based on the specifications given in the question.

```
class Time:
    def __init__(self, time):
        self.time = time
    def seconds_to_minutes(self):
        self.minutes = self.time // 60
        self.seconds = self.time % 60
        return f'{self.minutes} min {self.seconds} sec'

    def seconds_to_hours(self):
        self.seconds_to_minutes()
        self.hours = self.minutes // 60
        self.minutes = self.minutes % 60
        return f'{self.hours} hrs {self.minutes} min {self.seconds} sec'

    def seconds_to_days(self):
        self.seconds_to_hours()
        self.days = self.hours // 24
        self.hours = self.hours % 24
        return f'{self.days} days {self.hours} hrs {self.minutes} min {self.seconds}
sec'
```

Week 12

PPA1

Write a function named `primes_galore` that accepts a list `L` of non-negative integers as argument and returns the number of primes that are located at prime indices in `L`. For example:

L = [1, 3, 11, 18, 17, 23, 6, 8, 10]
The prime indices in the list are 2, 3, 5, 72,3,5,7. Of these, there are prime numbers at the indices 22 and 55. Therefore, the function should return the value 22 in this case.

```
defprimes_galore(L):
'''
    Argument:
        L: list of integers
    Return:
        result: integer
  '''
```

You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def is_prime(x):
    if x <= 1:
        return False
    for d in range(2, int(x ** 0.5) + 1):
        if x % d == 0:
            return False
    return True

def primes_galore(L):
    count = 0
    for i in range(len(L)):
        ifis_prime(i) and is_prime(L[i]):
            count += 1
    return count
```

PPA 2

Accept two positive integers a*a* and b*b* as arguments and print a rectangular pattern that has a*a* lines. The first and last line should have b*b* cirles, all other lines should have exactly two circles that are aligned with the two ends of the rectangle. You can assume that a, b \geq 2*a*,*b*≥2 for all test cases. A circle should be represented by the small letter "o".

Test cases (1) and (3) are misleading. Due to a formatting issue, the spaces are not represented properly. This is how you have to print it.

Test-Case-1

```
oooo
o  o
oooo
```

Test-Case-3

```
ooooooo
o    o
o    o
ooooooo
```

```
def rectangle(a, b):
    for i in range(1, a + 1):
        if i == 1 or i == a:
            print('o' * b)
        else:
            print('o', end = '')
            print(' ' * (b - 2) + 'o')

a = int(input())
b = int(input())
rectangle(a, b)
```

PPA 3

A square metal plate in 2D space is the setup we are going to work with. The spatial extent of the metal plate is given by:

0 \leq x, y \leq 50≤*x*,*y*≤5

The temperature at any point (x, y)(*x*,*y*) on the plate is given by the following equation. The temperature is measured in Celsius and can be negative:

f(x, y) = 30 + x^2 + y^2 -3x - 4y*f*(*x*,*y*)=30+*x*2+*y*2−3*x*−4*y*

A micro-organism lives on the surface of the metal plate. It occupies only those points on the plate where both the coordinates are integers. The organism cannot survive in high temperatures and instinctively moves to regions of low temperature that are less or equal to a threshold T*T*. If no such region is found, it can't survive. The terms high and low are used in a relative sense and should be compared with respect to the threshold.

Write a function named `survival` that accepts the value of T*T* as argument and returns `True` if the organism can survive on the metal plate, and `False` otherwise.

```
defsurvival(T):
'''
    Argument:
        T: integer
    Return:
        result: bool
  '''
```

You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def f(x, y):
    return30 + x ** 2 + y ** 2 - 3 * x - 4 * y

def survival(T):
    for x in range(6):
        for y in range(6):
            iff(x, y) <= T:
                return True
    return False
```

PPA 4

Write a function named `std_dev` that accepts a list of real numbers X*X* as argument. It should return the standard deviation of the points given by the following formula:

\sigma = \sqrt{\frac{\sum \limits_{i=0}^{n - 1} (X_i-\bar{X})^2}{n-1}}σ=*n*−1*i*=0∑*n*−1(*Xi*−*X*ˉ)2

Here, X_i*Xi* refers to the element X[i]*X*[*i*] in the list and \bar{X}*X*ˉ refers to the arithmetic mean of the numbers in X*X*. Try to use list-comprehension wherever possible. However, we won't be evaluating you on this.

```
defstd_dev(X):
'''
```

```
    Argument:
        X: list of integers
    Return:
        sigma: float
    '''
```

You do not have to accept the input from the user or print output to the console. You just have to write the function definition.

```
def std_dev(X):
    n = len(X)
    mean = sum(X) / n
    sigma = pow(sum([(x - mean) ** 2for x in X]) / (n - 1), 0.5)
    return sigma
X = [float(x) for x in input().split(',')]
sigma = std_dev(X)
print(f'{sigma:.2f}')
```

PPA 5

`word` is a string that contains one or more parentheses of the following types: "{ }", "[ ]", "( )". The string is said to be balanced if all the following conditions are satisfied. When read from left to right:

- Number of opening parentheses of a given type is equal to the number of closing parentheses of the same type.
- An opening parenthesis cannot be immediately followed by a closing parenthesis of a different type.
- Every opening parenthesis should be eventually closed by a closing parenthesis of the same type.

Write a function named `balanced` that accepts the string `word` as argument. Return `True` if the string is balanced, and `False` otherwise. You can assume that the string doesn't contain any characters other than parentheses.

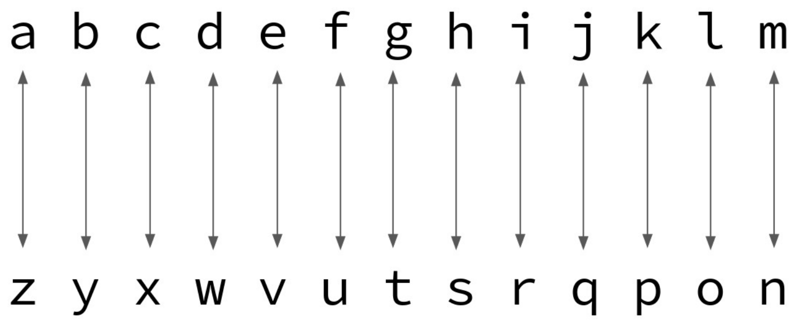
```
defbalanced(word):
'''
    Argument:
        word: string
    Return:
        result: bool
    '''
```

You do not have to accept the input from the user or print output to the console. You just have to write the function definition.

```
def matching(string):
    O = '([{'
    C = ')]}'
    R = [ ]
    forchar in string:
        ifchar in O:
            R.append(char)
        else:
            iflen(R) == 0:
                return False
            ind = C.index(char)
            ifR[-1] != O[ind]:
                return False
            else:
                R.pop()
    returnlen(R) == 0
```

PPA 6

Accept a string of lowercase letters from the user and encrypt it using the following image:



Each letter in the string that appears in the the upper half should be replaced with the corresponding letter in the lower half and vice versa. Print the encrypted string as output.

```
word = input()

letters = 'abcdefghijklmnopqrstuvwxyz'

out = ''
forchar in word:
    ind = letters.index(char)
    out = out + letters[-ind - 1]

print(out)
```

PPA 7

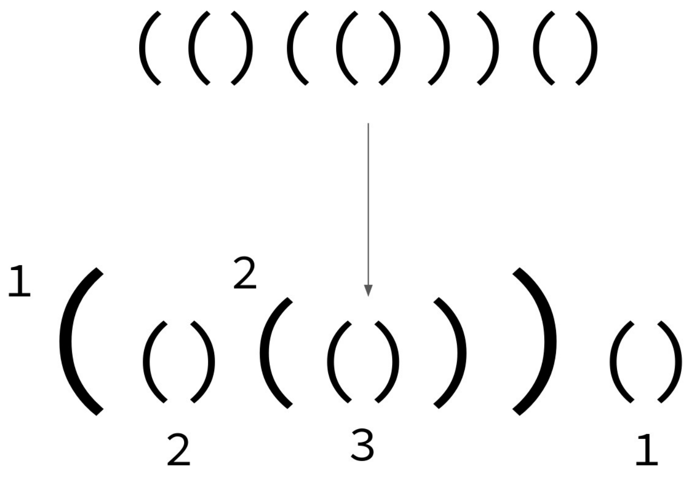
A clockwise rotation of a list consists of taking the last element and moving it to the beginning of the list. For instance, if we rotate the list [1, 2, 3, 4, 5], we get [5, 1, 2, 3, 4]. If we rotate it again, we get [4, 5, 1, 2, 3]. Your task is to perform k*k* rotations of a list.

The first line of input contains a non-empty sequence of comma-separated integers. The second line of input is a positive integer kk. Perform kk rotations of the list and print it as a sequence of comma-separated integers.

```
L = list(eval(input()))
k = int(input())

for _ in range(k):
    L.insert(0, L.pop())

print(','.join([str(x) for x in L]))
```

PPA 8

Consider the problem about balanced expressions discussed in PPA-5. We have a balanced expression (string) that has only the flower brackets: '( )'. We can recursively define a concept called nesting depth for each pair of opening and closing brackets. The nesting depth of a pair that lies within another pair is one more than the nesting depth of the pair that immediately englobes it. For a pair that is not surrounded by any other pair, the nesting depth is 11.



Write a function named `depth` that accepts a balanced expression (string) as argument. It should return the maximum nesting depth in this expression.

```
def depth(exp):
    '''
    Argument:
        exp: string
    Return:
        result: integer
    '''
```

You do not have to accept the input from the user or print output to the console. You just have to write the function definition.

```
def depth(exp):
    m_depth = 0

    k = 0
    for c in exp:
        if c == '(':
            k += 1
        elif c == ')':
            k -= 1

        if k > m_depth:
            m_depth = k

    return m_depth
```

PPA 9

The Pearson correlation coefficient is a measure of association between two sets of data. In this problem, the data shall be represented as vectors (lists) X*X* and Y*Y*. The formula for computing the coefficient is given below:

\rho(X, Y) = \frac{\sum \limits_{i=0}^{n-1}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum \limits_{i=0}^{n-1}(X_i - \bar{X})^2}\cdot\sqrt{\sum \limits_{i=0}^{n-1}(Y_i - \bar{Y})^2}}*ρ*(*X*,*Y*)=*i*=0∑*n*−1(*Xi*−*X*¯)2⋅*i*=0∑*n*−1(*Yi*−*Y*¯)2*i*=0∑*n*−1(*Xi*−*X*¯)(*Yi*−*Y*¯)

X_i*Xi* corresponds to the element X[0]*X*[0] in the list X*X*. We have used zero-indexing here.

Write a function named `pearson` that accepts two vectors X*X* and Y*Y* as arguments and returns the Pearson correlation coefficient between them.
To help you with the computation, some functions have already been defined in the prefix code. You can write the entire function by just using the pre-defined functions. This is an important exercise that will help you in future projects when you start working with libraries.

```
defpearson(X,Y):
'''
    Arguments:
        X: list of float
        Y: list of float
    Return:
        result: float
  '''
```

You do not have to accept the input from the user or print output to the console. You just have to write the function definition.

```
def f(P):
    mean = sum(P) / len(P)
    return [p - mean for p in P]

def g(P, Q):
    returnsum(P[i] * Q[i] for i in range(len(P)))
```

```
def h(x):
    return x ** 0.5
def pearson(X, Y):
    X_norm, Y_norm = f(X), f(Y)
    num = g(X_norm, Y_norm)
    den = h(g(X_norm, X_norm) * g(Y_norm, Y_norm))
    return num / den
X = [float(x) for x in input().split()]
Y = [float(y) for y in input().split()]
print(f'{pearson(X, Y):.2f}')
```

PPA 10

A queue at a fast-food restaurant operates in the following manner. The queue is a single line with its head at the left-end and tail at the right-end. A person can exit the queue only at the head (left) and join the queue only at the tail (right). At the point of entry, each person is assigned a unique token.
You have been asked to manage the queue. The following operations are permitted on it:

Operation	Meaning
JOIN,token	Add a person with this token number to the tail of the queue.
LEAVE	Remove the person from the head of the queue.
MOVE,token,HEAD	Move the person with this token number to the head of the queue.
MOVE,token,TAIL	Move the person with this token number to the tail of the queue.
PRINT	Print the queue as a sequence of comma-separated token numbers. The head comes to the left, tail to the right.

The queue is empty to begin with. Keep accepting an operation as input from the user until the string "END" is entered.

```
Q = [ ]

op = input().split(',')
whileop[0] != 'END':
    ifop[0] == 'JOIN':
        Q.append(op[1])
    elif op[0] == 'LEAVE':
        Q.pop(0)
    elif op[0] == 'MOVE':
        member = op[1]
        Q.remove(member)
        ifop[2] == 'TAIL':
            Q.insert(len(Q), member)
        else:
            Q.insert(0, member)
    ifop[0] == 'PRINT':
        print(','.join(Q))
    op = input().split(',')
```

PPA 11

You are given a CSV file that contains the marks of students in various subjects. The first line of the file is the header. The first column of the file holds the names of students. All other columns correspond to some subject. Sample file:

```
Name,Physics,Biology
Ram,83,48
Nitin,64,31
Mayur,86,51
```

Your task is to read the file and store the details in the form of a dictionary. The keys of the dictionary are the column names. The value corresponding to a key is the column in the file represented as a list. For example:

```
{
"Biology":[
48,
31,
51
],
"Name":[
"Ram",
"Nitin",
"Mayur"
],
"Physics":[
83,
64,
86
]
}
```

For any list the elements should be appended to it in the same order as they appear in the corresponding column in the file.

Write a function named `file_to_dict` that accepts the filename as argument. It should return the dictionary in the structure mentioned above.

```
deffile_to_dict(fname):
'''
    Argument:
        fname: string
    Return:
        result: dictionary
  '''
```

---

(1) You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

(2) The number of columns will keep varying across test cases. The first column of each file is guaranteed to be "Name".

(3) Make sure that all subject marks are of type `int`.

```python
def file_to_dict(fname):
    f = open(fname, 'r')
    D = dict()
    # Here we need to store the header in a list
    header = f.readline().strip().split(',')
    # Add header names as keys in the dictionary
    for col in header:
        D[col] = [ ]
    # Iterate through each line in the file
    for line in f:
        cols = line.strip().split(',')
        for i in range(len(cols)):
            # Get the column name
            col_name = header[i]
            # Make sure that the value is integer for all marks
            val = int(cols[i]) if col_name != 'Name'elsecols[i]
            D[col_name].append(val)
    f.close()
    return D
```

PPA 12

You are given a CSV file that contains the marks of students in various subjects. The first line of the file is the header. The first column of the file holds the names of students. All other columns correspond to some subject. Sample file:

```
Name,CompSci,Architecture,Physics
Ram,80,74,85
Nitin,94,44,98
Mayur,45,40,88
Usha,63,36,56
Keerthi,72,59,69
Fatima,56,52,48
```

Your task is to read the file and store the details as a list of dictionaries. Each element in the list corresponds to one row of the file. If the file has nn lines, the list should have n - 1$n-1$ elements. For example, the first entry in the list corresponds to the second row in the file:

```json
{
"Architecture":74,
"CompSci":80,
"Name":"Ram",
"Physics":85
}
```

The elements should be appended to the list in the order in which they appear in the file.

Write a function named `file_to_list` that accepts the filename as argument. It should return the list in the format given above.

```
deffile_to_list(fname):
'''
    Argument:
      fname: string
    Return:
      result: list of dicts
  '''
```

(1) You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

(2) The number of columns will keep varying across test cases. The first column of each file is guaranteed to be "Name".

(3) Make sure that the subject marks is of type `int`.

```
def file_to_list(fname):
    f = open(fname, 'r')
    # We have to store the header in a variable
    header = f.readline().strip().split(',')
    # each element of L will be one row of the file
    L = [ ]
    for line in f:
        # The dict D will correspond to one row of the file
        D = dict()
        row = line.strip().split(',')
        for i in range(len(row)):
            val = int(row[i]) ifheader[i] != 'Name'elserow[i]
            D[header[i]] = val
        L.append(D)
    f.close()
    return L
```

PPA 13

Gurunath is a popular store inside IIT Madras. Among other things, it sells stationary items. The owner of the stores gives you the list of transactions that have happened in a day. Each transcation comes with a unique transaction ID. He wants to estaimte the cost of each transaction. Can you help him out?

The list `trans` is a list of transactions that happened at the shop in a given day. Each element of this list is a dictionary. The details of one such transaction is given below:

```
{
'TID':'Gurunath_8528',
'Items':[
{'Name':'Notebook','Price':50,'Qty':4},
{'Name':'Pencil','Price':10,'Qty':1},
{'Name':'Eraser','Price':15,'Qty':1},
{'Name':'File','Price':80,'Qty':1}
]
}
```

Does this remind you of the shopping bills dataset from CT?

Write a function named `get_summary` that accepts the list `trans` as argument. It should return a list of dictionaries. Each dictionary should have two keys: "TID" and "Cost". For example, one of the elements of the list would be as follows:

```
{
"Cost":305,
"TID":"Gurunath_8528"
}
```

The computation of the cost is given below:

50 \times 4 + 10 \times 1 + 15 \times 1 + 80 \times 1 = 30550×4+10×1+15×1+80×1=305

```
defget_summary(trans):
'''
    Argument:
        trans: list of dicts
    Return:
        result: list of dicts
  '''
```

---

(1) The order of elements in the returned list should be the same as the order in the input list. That is, the i^{th}*ith* element in the returned list should correspond to the transaction cost of the i^{th}*ith* element in `trans`.
(2) You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def get_summary(trans):
    summary = list()
    # T corresponds to one transaction
    for T in trans:
        D = {'TID': T['TID']}
        val = 0
        # We are computing the cost of that transaction
        # For each item, the cost is its price times the quantity
        # We need to sum the cost of all items to get the total cost
        for item in T['Items']:
            val = val + item['Price'] * item['Qty']
        D['Cost'] = val
        summary.append(D)
    return summary
```

PPA 14

The `scores_dataset` is a list of dictionaries one of whose entries is given below for your reference:

```
{'SeqNo':1,'Name':'Devika','Gender':'F','City':'Bengaluru',
'Mathematics':85,'Physics':100,'Chemistry':79,'Biology':75,
'Computer Science':88,'History':60,'Civics':88,'Philosophy':95}
```

An institution decides to allow students to create student groups for each subject where students with similar marks can help each other. But it draws up a set of constraints for creating student groups:

- A group should be associated with a particular `subject`.
- The difference between the scores of any two students in the group should be at most `mark_limit`.

It follows that multiple groups can be formed for a given subject.

Write a function called `crowded_group` that accepts three arguments as input:

- `scores_dataset`
- `subject`
- `mark_limit`

Return the size of the largest possible group in this `subject` with the given `mark_limit`.

```
defcrowded_group(scores_dataset,subject,mark_limit):
'''
    Arguments:
      scores_dataset: list of dicts
      subject: string
      mark_limit: integer
    Return:
      result: integer
  '''
```

You do not have to accept input from the user or print the output to the console. You just have to write the function definition.

```
def crowded_group(scores, sub, mark_limit):
    # Extract the scores of all students in this subject
    L = [ ]
    for st in scores:
        L.append(st[sub])
    # Sort the scores
    L.sort()
    # Initialize the group-size to 1
    gsize = 1
    # We assume that L[i] is the least marks in the group
    for i in range(len(L)):
        for j in range(i, len(L)):
            # If L[j] - L[i] <= mark_limit, then j can be included in the group
            ifL[j] - L[i] <= mark_limit:
                # j - i + 1 is the size of the group now
                if j - i + 1> gsize:
                    gsize = j - i + 1
    return gsize
```

PPA 15

The `scores_dataset` is a list of dictionaries one of whose entries is given below for your reference:

```
{'SeqNo':1,'Name':'Devika','Gender':'F','City':'Bengaluru',
'Mathematics':85,'Physics':100,'Chemistry':79,'Biology':75,
'Computer Science':88,'History':60,'Civics':88,'Philosophy':95}
```

A student *XX* can mentor another student *YY* in `subject` if the following condition is satisfied:

10 \leq X.\text{subject} - Y.\text{subject} \leq 2010≤*X*.subject−*Y*.subject≤20

Write a function named `mentors` that accepts the following arguments:

- `scores_dataset`
- `subject`

The function should return a dictionary having the following structure:

- key: `SeqNo` of a student
- value: list of `SeqNo` of students who can be mentored by the above student

---

```
defmentors(scores_dataset,subject):
'''
    Arguments:
      scores: list of dicts
      subject: string
    Return:
      result: dict
        key: integer (SeqNo)
        value: list of integers (SeqNo)
  '''
```

---

(1) You do not have to accept input from the user or print output to the console.

(2) We randomly sample five elements from the dictionary that you return and print them to the console. You don't have to worry about the order in which the `SeqNo` are entered into any list.

(3) Note that a student cannot mentor himself! Also, if a student cannot mentor anyone, then the list should be empty.

```
def mentors(scores, subject):
    # Get the scores and seq-no of all students in this subject
    L = [ ]
    for S in scores:
        L.append((S['SeqNo'], S[subject]))
    D = dict()
    for i in range(len(L)):
        # L[i][0] is the sequence number
        D[L[i][0]] = [ ]
        for j in range(len(L)):
            if i == j:
                continue
            # Check if i can mentor j
            if10<= L[i][1] - L[j][1] <= 20:
                D[L[i][0]].append(L[j][0])
    return D
```

PPA 16

`players` is a dictionary that has the details of the sports played by some people. The keys are names of the people. The value corresponding to a key is another dictionary. The keys of the inner dictionary are the names of the sports, and the values are Boolean values which represent whether the person plays a sport or not. For example:

```
players={
'Karthik':{
'Tennis':True,
'Badminton':True,
'Cricket':False
},
'Rahul':{
'Tennis':False,
'Badminton':False
'Cricket':True
}
}
```

Here, Karthik plays tennis and badminton, but not cricket.

Write a function named `exactly_two` that accepts the dictionary `players` as argument and returns the set of all players who play exactly two sports.

```
defexactly_two(players):
'''
    Argument:
        players: dict of dicts
    Return:
        result: set of strings
  '''
```

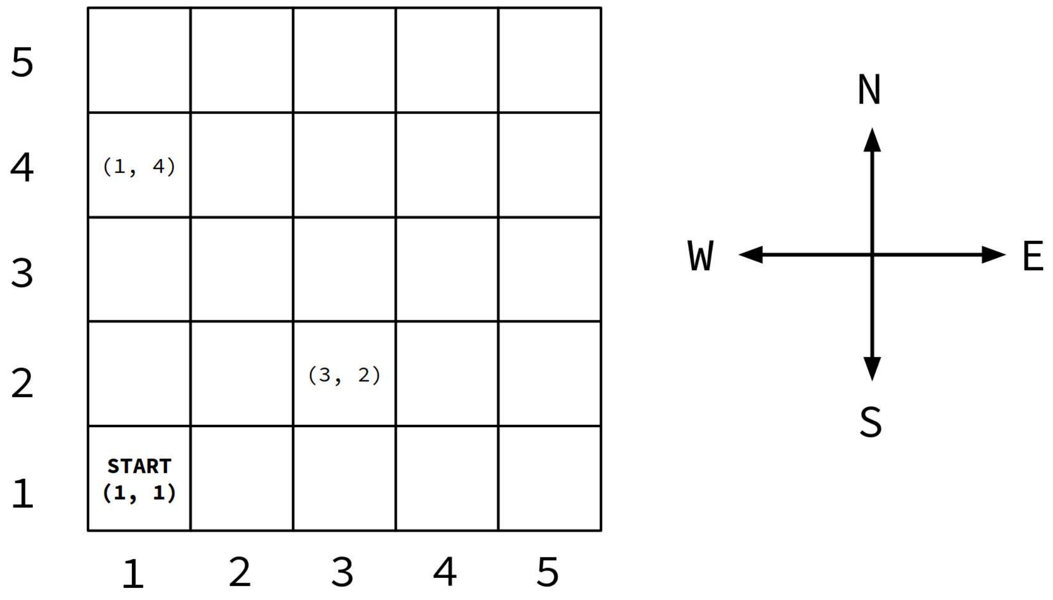
You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def exactly_two(players):
    pairs = [('Tennis', 'Cricket', 'Badminton'),
             ('Cricket', 'Badminton', 'Tennis'),
             ('Badminton', 'Tennis', 'Cricket')]
    names = set()
    for player in players:
        for s1, s2, s3 in pairs:
            if (players[player][s1] and
                players[player][s2] and
                (not players[player][s3])):
```

```
            names.add(player)
    return names
```

PPA 17

Consider a grid-world of size n \times n$n \times n$. You are at the bottom-left corner, at the cell (1, 1)(1,1), to start with. A sample grid-world for the case of n = 5n=5 is given below for your reference:



You can move one step at a time in any one of the four directions: "North", "East", "West", "South". At every cell of the grid, all four moves are possible. The only catch is that if you make a move that will take you out of the grid at a border cell, you will stay put at the same cell. Concretely, if you are at the cell (1, 4)(1,4) and try to move west, you will remain at the same cell. Or if you are at (3, 1)(3,1) and try to move south, you will remain there.

Write a function named `final` that accepts a positive integer nn and a sequence of `moves` (string) as arguments and returns the final position where you would end up in an n \times n$n \times n$ grid-world if you start at (1, 1)(1,1). You must return a tuple of size 2, where the first element is the x-coordinate and the second is the y-coordinate.

```
deffinal(n,moves):
'''
    Argument:
        n: int
        moves: string
    Return:
        (x, y): tuple
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def final(n, moves):
    pos = [1, 1]
    for m in moves:
        if m == 'N':
            ifpos[1] < n:
                pos[1] += 1
        if m == 'S':
            ifpos[1] >1:
                pos[1] -= 1
        if m == 'E':
            ifpos[0] < n:
                pos[0] += 1
        if m == 'W':
            ifpos[0] >1:
                pos[0] -= 1
        assert m in 'NEWS'
    returnpos[0], pos[1]
```

PPA 18

Write the following functions:

(1) Write a function named `list_to_dict` that accepts a nested list `L` as argument and returns a dictionary `D` that has the following structure:

- keys of D are integers in the range [0, \text{len}(L) - 1][0,len(L)−1], endpoints inclusive.
- values corresponding to key ii should be L[i]$L[i]$.

(2) Write a function named `dict_to_list` that accepts a dictionary `D` as argument and returns a nested list `L`. The dictionary `D` has the following structure:

- keys of D are integers in the range [0, \text{len}(L) - 1][0,len(L)−1], endpoints inclusive.
- values corresponding to key ii is a list

The element L[i]$L[i]$ of the returned list should be the value corresponding to the key ii in the dictionary.

```
deflist_to_dict(L):
'''
    Argument:
        L: list of lists
    Return:
        D: dict
            key: int
            value: list
  '''

defdict_to_list(D):
'''
    Argument:
        D: dict
```

```
            key: int
            value: list
        Return:
            L: list of lists
    '''
```

You do not have to accept input from the user or print output to the console. You just have to write the definition for both the functions.

```
def list_to_dict(L):
    return {i: L[i] for i in range(len(L))}

def dict_to_list(D):
    return [D[key] for key in range(len(D))]
```

PPA 19

In spreadsheets, columns are labeled as follows:

Label	Number
A	11
B	22
Z	2626
AA	2727
AB	2828
AZ	5252
BA	5353
ZZ	702702
AAA	703703
ZZZ	1827818278

A is the first column, B is the second column, Z is the 26th column and so on. Using the table given above, deduce the mapping between column labels and their corresponding numbers. Accept the column label as input and print the corresponding column number as output.

```
def lmap(letter):
    letter_map = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    return letter_map.index(letter) + 1

label = input()
n = len(label)

def colnum(label):
    n = len(label)
    val = lmap(label[0])
    if n == 1:
        return val
    return val * 26 ** (n - 1) + colnum(label[1: ])

print(colnum(label))
```

PPA 20

A valid password should satisfy all the following conditions:

- Length of password should be greater than 77.
- The first character should be an upper case letter.
- The remaining characters should be either numbers or letters or a combination of both. That is, they should all be alpha-numeric characters.

Write a class `Profile` with the following specification:

Attributes

- `username`: string
- `password`: string
- `old_passwords`: a list that holds the history of the user's passwords. The last item of this list is the user's current password.

Methods

- `__init__(self, username, password)`: Assign the argument `username` to the corresponding attribute.
- If the password is valid, then initialize the attribute `password` appropriately and initialize the list `old_passwords` with this password as the only element. Use the method `check_password` for validating the password.
- **If it is not a valid password, then initialize the corresponding attribute to `None` and set `old_passwords` to an empty list.**
- **`check_password(self, password)`: Accepts `password` as argument and returns `True` if it is a valid password and `False` otherwise.**
- **`change_password(self, password)`: Accept `password` as argument and update the user's current password and the append this password to `old_passwords` provided the both the conditions given below are satisfied:**
- **`password` should be a valid password. Use the method `check_password` for validation.**
- `password` should *not* be found in the list `old_passwords`.
If even one of these conditions is not satisfied, just leave the current password as it is.

- `login(self, username, password)`: Accept `username` and `password` as arguments and print a message "WELCOME" if the login details match the current values. Otherwise, print the message "INVALID".

You do not have to accept input from the user or print the output to the console. You just have to write the definition of the class.

```
class Profile:
    def __init__(self, username, password):
        self.username = username
        if self.check_password(password):
            self.password = password
            self.old_passwords = [password]
        else:
            self.password = None
            self.old_passwords = [ ]
    def check_password(self, password):
        iflen(password) <= 7:
            return False
        if not('A'<= password[0] <= 'Z'):
            return False
        forchar in password:
            if not char.isalnum():
                return False
        return True

    def change_password(self, password):
        if (self.check_password(password) and
            password not in self.old_passwords):
            self.password = password
            self.old_passwords.append(password)

    def login(self, username, password):
        if (self.username == username and
            self.password == password):
            print('WELCOME')
        else:
            print('INVALID')
```

PT 1.1

Print the following pattern. There is exactly one space between any two consecutive numbers on any line. There are no spaces at the end of any line.

```
1 2 1
1 2 3 2 1
1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
```

```
print('1 2 1')
print('1 2 3 2 1')
print('1 2 3 4 3 2 1')
print('1 2 3 4 5 4 3 2 1')
```

PT 1.2

A simple algorithm has to be designed to find out whether a student belongs to the Data Science branch or not. The input will be a student's roll number, which is of the form BR18B0000. Here, BR represents the branch code, 18 represents the year of joining, B represents the education level and 0000 represents the specific identification given to the student of that batch. The branch code for Data Science is DS. Print True if the student belongs to Data Science branch and False otherwise.

```
s = input()
if s[0]=='D' and s[1]=='S':
    print("True")
else:
    print("False")
```

PT 1.3

The police are trying to track a criminal based on the evidence available at a crime site. Their main clue is a vehicle's damaged number plate. Only the string TN07 is visible. The format of the registration number is AA00AA00, where the first two letters are alphabets, next two are numbers, next two are again alphabets followed by two numbers at the end. A number plate is picked from a database of registration numbers and is given to you as input. Your task is to determine if this could belong to the criminal or not. Print True if the number plate contains TN07 and False otherwise.

```
n=input()
if (n[0:4]=='TN07') or (n[4:8]=='TN07'):
    print('True')
else:
    print('False')
```

PT 1.4

Accept two integers a and b as input and print the absolute difference of both the numbers. For example, if a = 9, b = 8, then the absolute difference is 9 - 8 = 1. So, your output should be 1. You should be able to solve this problem using the concepts covered in this week.

```
a=int(input())
b=int(input())
print(abs(a-b))
```

PT 1.5

You are given a string and two non-negative integers as input. The two integers specify the start and end indices of a substring in the given string. Create a new string by replicating the substring a minimum number of times so that the resulting string is longer than the input string. The input parameters are the string, start index of the substring and the end index of substring (endpoints inclusive) each on a different line.

```
n=input()
a=int(input())
b=int(input())
x=n[a:b+1]
y=len(n)
ans=""
while(y>=len(ans)):
ans+=x

print(ans)
```

PT 2.1

A class teacher has decided to split her entire class into four groups, namely Sapphire, Peridot, Ruby, and Emerald for sports competitions. For dividing the students into these four groups, she has followed the pattern given below:

Sapphire - 1, 5, 9, 13, 17, 21, ...
Peridot - 2, 6, 10, 14, 18, 22, ...
Ruby   - 3, 7, 11, 15, 19, 23, ...

Emerald - 4, 8, 12, 16, 20, 24, ...

All the students are represented by their roll numbers. Based on the above pattern, given the roll number as input, print the group the student belongs to. Note that the roll number can be any positive integer and not necessarily less than 25.

```
n=int(input())
for i in range(1,n+1,4):
    if n==i:
        print('Sapphire')
for j in range(2,n+1,4):
    if n==j:
        print('Peridot')
for k in range(3,n+1,4):
    if n==k:
        print('Ruby')
for l in range(4,n+1,4):
    if n==l:
        print('Emerald')
```

PT 2.2

A data science company wants to hire data scientists from IIT Madras. The company follows a certain criteria for selection: for a student to be selected, the number of backlogs should be at most 5 and the CGPA (Cumulative Grade Point Average) should be greater than 6. If the student does not fit the above criteria, then the student is not offered the job. If the student is selected, then the salary offered is equal to 5 times his/her CGPA (in lakhs).

Accept the number of backlogs (integer) and the CGPA (float) of the student as input. Your task is to determine if the student is selected or not. If the student is selected, then print the package. If not, then print the string `Not Selected`.

```
a=float(input())
b=float(input())
if a<=5 and b>6:
    print(5*b)
else:
print('Not Selected')
```

PT 2.3

A test match happened between Team A and Team B. The scores of the teams in both the innings are given as input in eight lines in the format given below. The first two lines represent the scores of Team A in the first innings and the next two lines represent the scores of Team A in the second innings. Likewise, the last four lines represent the scores of Team B in both the innings.

The numbers in 2nd, 4th, 6th, and 8th lines represent the wickets lost by the teams and the numbers in 1st, 3rd, 5th, and 7th represent the runs scored.

```
120
10
210
10
115
10
189
10
```

In the above example, team-A has scored 120 for the loss of 10 wickets in the first innings, and 210 for the loss of 10 wickets in the second innings. Team A plays first and Team B plays second. Your task is to determine the winner of the match.

Process to decide the outcome
Team A wins if and only if the sum of its scores in both the innings is greater than sum of the scores of Team B in both the innings AND Team B lost all the ten wickets in the second innings. Team B wins if the sum of its scores in both the innings is greater than sum of the scores of Team A in both the innings.
A match will end in a draw if the sum of scores in the two innings of both the teams are equal OR if `Team B` did not lose all the ten wickets in the second innings. If the match ends in a draw, then print `DRAW`.
Example

```
120
10
210
10
115
10
189
10
```

Example output

```
Team A
```

120 + 210 > 115 + 89 and Team B lost all 10 wickets in second innings, therefore Team A is the winner of the test match.

```
a=int(input())
b=int(input())
c=int(input())
d=int(input())
e=int(input())
f=int(input())
g=int(input())
h=int(input())
if a+c>e+g and h==10:
```

```
print('Team A')
elifa+c<e+g:
print('Team B')
else:
    print('DRAW')
```

PT 2.4

A word is said to be **perfect** if it satisfies all the following criteria:
(1) All the vowels (a,e,i,o,u) should be present in the word.

(2) Let the vowels be represented as$v_1=a,v_2=e,v_3=i,v_4=o,v_5=u$ in lexical order.

- If $i<j$, then the first appearance of v_i in the word should come before the first appearance of v_j.
- If$i<j$, then the count of v_i should be greater than or equal to the count of v_j.

Accept a word as input. Print `It is a perfect word.` if the word is perfect, else print `It is not a perfect word.`

```
s=input()
a='aeiou'
f1=True
for i in range(5):
    if a[i] not in s:
        f1=False
        break
if f1:
    if s.index('a')<s.index('e')<s.index('i')<s.index('o')<s.index('u'):
        if
s.count('a')>=s.count('e')>=s.count('i')>=s.count('o')>=s.count('u'):
print('It is a perfect word.')
        else:
print('It is not a perfect word.')
else:
print('It is not a perfect word.')
```

PT 2.5

Accept four integers as input and write a program to print these integers in non-decreasing order.

The input will be four integers in four lines. The output should be a single line with all the integers separated by a single space in non-decreasing order.

Note: There is no space after the fourth integer.

```
a=int(input())
b=int(input())
c=int(input())
d=int(input())
x=[a,b,c,d]
x.sort()
print(x[0],x[1],x[2],x[3])
```

PT 3.1

Accept a string as input and print PALINDROME if it is a palindrome, and NOT PALINDROME otherwise.

```
string=input()
if(string==string[::-1]):
      print("PALINDROME")
else:
print("NOT PALINDROME")
```

PT 3.2

Two integers are co-prime if the only divisor common to them is one. Accept two distinct positive integers as input in two different lines. Print Coprime if the two integers are co-prime, else print Not Coprime. Assume that both the integers are greater than two.

```
def are_coprime(a,b):

hcf = 1

    for i in range(1, a+1):
        if a%i==0 and b%i==0:
hcf = i

    return hcf == 1
first = int(input())
second = int(input())
if are_coprime(first, second):
    print('Coprime')
else:
print('Not Coprime')
```

PT 3.3

Accept a string as input, print Integer if the string is an integer, print Float if it a float, else print None.

```
n=input()
if '.'in n and n.count('.')==1:
    n=n.replace('.','')
    if n.isnumeric():
        print('Float')
    else:
        print('None')
elifn.isnumeric():
    print('Integer')
else:
    print('None')
```

PT 3.4

Multiple Select Questions (MSQ) could have more than one correct answer. The marks scored by a studentin a MSQ will be determined by the following conditions:

(1) If the question has c correct options, each individual correct option carries = marks

(2) If a student selects any of the wrong options, the marks awarded for the question will be 0.

Calculate the marks obtained by the student and print this as float value.

The input contains four lines.

(1) First line is the number of marks for the question.

(2) Second line contains the number of options for the question.

(3) Third line is a comma-separated sequence of correct options for this question.

(4) Fourth line is a comma-separated sequence of options given by the student.

Write a program to print the number of marks scored by a student.

Note: Options are numbered using positive integers in the range [1, 9], endpoints inclusive. A question willhave at most nine options. The number of marks and the correct options will always be integers.

If the question has five options in total, then the options will be numbered as 1,2,3,4,5.

```
marks = int(input())
options = int(input())
correct_options = input().split(',')
answered_options = input().split(',')
c = 0
for i in answered_options:
    if i in correct_options:
        c += 1
    else:
        c = 0
        break
marks_per_option = marks / len(correct_options)
```

```
total_marks = marks_per_option * c
print(total_marks)
```

PT 3.5

Consider two non-empty strings `a` and `b` of lengths n_1 and n_2 respectively, which contain only numbers as their characters. Both the strings are in ascending order, that is $a[i] \leq a[j]$ for $0 \leq i < j < n_1$. The same holds true for `b`. You need to merge both the strings into one string of length $n_1 + n_2$ such that all the characters of this combined string are also in ascending order.
Accept `a` and `b` as input and print this merged string as output.

```
n=input()
s=input()
t=s+n
m=''
w='0123456789'
for i in range(len(w)):
  for j in range(len(t)):
    if t[j]==w[i]:
      m+=w[i]
print(m)
```

PT 4.1

You are given a list of strings. Where each string contains two integers that are comma separated. For each of the string, you need to check whether the given two integers in string are co-prime or not. You need to print list of values as YES or NO for each value of the list separated by comma. Print YES if the pair of integers are co-prime else print NO.

```
n = int(input())
A=[]
for i in range(n):
A.append(input().split(','))
    for j in range(2):
        A[-1][j] = int(A[-1][j])
def factors(n):
    l=[]
    for i in range(1,n+1):
        if n%i==0:
l.append(i)
    l=set(l)
    return l
x=[]
for i in A:
    if factors(i[0])&factors(i[1])=={1}:
x.append('YES')
    else:
```

```
x.append('NO')
print(*x,sep=',')
```

PT 4.2

The first line of input contains a positive integer *n*. The second line of input contains a sequence of comma-separated positive integers. Print the minimum number of terms that need to be picked up from the sequence so that the sum of these terms is greater than or equal to *n*. If no such minimum number exists, then print the string `None`.

---

(1) For example, if the input is 100 and the sequence is 10,88,3,4,99, then the minimum number of terms we need to pick up is 2 so that their sum is greater than or equal to 100.
(2) If the input is 99 and the sequence is 10,20,5,18,17, then we can never get a sum that is greater than or equal to 99, no matter how many numbers we pick up. So, the output here is `None`.

```
a=int(input())
b=input().split(',')
total=0
count=0

for i in range (len(b)):
    for i in range(len(b)-1):
        if int(b[i])<int(b[i+1]):
            b[i],b[i+1]=b[i+1],b[i]

for i in range(len(b)):
    total =total+int(b[i])
    count+=1
    if total >=a:
        print(count)
        break
else:
    print('None')
```

PT 4.3

Two strings are said to be equivalent if either string can be obtained by rearranging the characters of the other string. For example, good and odog are equivalent. But apple and lape are not equivalent. Accept two strings as input. Print Equivalent if both the strings are equivalent and Not Equivalent otherwise.

```
s1=input()
s2=input()
flag=True
if len(s1)==len(s2):
    for i in s1:
        if s1.count(i)!=s2.count(i):
print('Not Equivalent')
            flag=False
            break
    if flag:
        print('Equivalent')
else:
print('Not Equivalent')
```

PT 4.4

A is a square matrix of size $n \times n$ that is given to you. m is a positive integer that is also given to you. Print the value of A^m.

```
n = int(input())
m = int(input())
A=[]
for i in range(n):
A.append(input().split(','))
    for j in range(n):
        A[-1][j] = int(A[-1][j])
B=A
for x in range(m-1):
    t=[]
    for i in range(n):
        y=[]
        for j in range(n):
            c=0
for k in range(n):
                c+=A[i][k]*B[k][j]
y.append(c)
t.append(y)
    B=t
print(B)
```

PT 5.1

A function $f(n)$ is defined as $f(n)= 1*3*5*7*\cdots*n-1/2*4*6*8*10*\cdots*n$ if n is even, and$f(n)= 1*3*5*7*\cdots*n/2*4*6*8*10*\cdots*n-1$ if n is odd. Assume that n is a positive integer. Given a value of n as input, write a function named `func` to return the value of $f(n)$. Value of `f(1)` is 1.

```
def func(n):
    if n==1:
        return 1
    c=1
    for i in range(1,n+1):
        if i%2==1:
            c*=i
        else:
            c/=i
    return c
```

PT 5.2

In a portal login website, you are asked to write a function `get_password_strength` to decide the strength of a password. The strength is decided based on the total score of the password, Use following conditions:
1) If password has length greater than 7 then score increases by one point.
2) If password has at least one upper case and one lower case alphabets score increases by one point.
3) If password has at least one number and no consecutive numbers like `12` or `234` then score increases by one point.
4) If password has at least one special character(any character other than numbers and alphabets) then score increases by one point.
5) If password contains username, then it is invalid password.
If the password has score of four points, three points, two points, or one point then print `Very Strong`, `Strong`, `Moderate`, or `Weak` respectively. If the password is invalid, then print `PASSWORD SHOULD NOT CONTAIN USERNAME` and If the score is zero, then print `Use a different password`.The arguments to the function are `username` and `password` which are already defined.

```
def get_password_strength(username,password):
    c=0
    capital='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
c_flag=False
    small='abcdefghijklmnopqrstuvwxyz'
s_flag=False
num='0123456789'
n_flag=False
nc_flag=True
    if username in password:
print('PASSWORD SHOULD NOT CONTAIN USERNAME')
        return
    if len(password)>7:
        c=c+1
    for i in range(len(password)):
        if password[i] in small:
s_flag=True
elif password[i] in capital:
c_flag=True
elif password[i] in num:
n_flag=True
        if n_flag:
          if i!=(len(password)-1):
            if num[((num.index(password[i]))+1)]==password[i+1]:
nc_flag=False
    if c_flag and s_flag :
        c+=1
    if n_flag and nc_flag:
        c+=1
    if not password.isalnum():
        c+=1
    if c==4:
```

```
print('Very Strong')
        return
    if c==3:
        print('Strong')
        return
    if c==2:
        print('Moderate')
        return
    if c==1:
        print('Weak')
        return
    if c==0:
print('Use a different password')
        return
```

ProgQuiz-M1

A data entry operator has a faulty keyboard. The keys 0 and 1 are very unreliable. Sometimes they work, sometimes they don't. While entering phone numbers into a database, the operator uses the letter 'l' as a replacement for 1 and 'o' as a replacement for 0 whenever these binary digits let him down. Both 'l' and 'o' are in lower case.

Accept a ten-digit number as input. Find the number of places where the numbers 0 and 1 have been replaced by letters. If there are no such replacements, print the string `No mistakes`. If not, print the number of mistakes (replacements) and in the next line, print the correct phone number.

```
n=input()
o=n.count('o')
l=n.count('l')
if o!=0:
    for j in range(o):
        n=n.replace('o','0')
if l!=0:
    for j in range(l):
        n=n.replace('l','1')
if o+l==0:
print('No mistakes')
else:
    print(o+l,'mistakes')
    print(n)
```

ProgQuiz-M2

A sequence of integers of even length is said to be `left-heavy` if the sum of the terms in the left-half of the sequence is greater than the sum of the terms in the right half. It is termed `right-heavy` if the sum of the second half is greater than the first half. It is said to be `balanced` if both the sums are equal.

Accept a sequence of comma-separated integers as input. Determine if the sequence is `left-heavy`, `right-heavy` or `balanced` and print this as the output.

```
l=input().split(',')
n=(len(l)//2)
left=0
right=0
for i in range(n):
    left+=int(l[i])
    right+=int(l[-(i+1)])
if left>right:
    print('left-heavy')
elif left<right:
    print('right-heavy')
elif left==right:
    print('balanced')
```

ProgQuiz-M3

A square matrix M is said to be:

- diagonal: if the entries outside the main-diagonal are all zeros
- scalar: if it is a diagonal matrix, all whose of diagonal elements are equal
- identity: if it is a scalar matrix, all of whose diagonal elements are equal to 1

Accept a matrix *M* as input from the console. The first line of input will have *n*, the number of rows in the matrix. Each of the next *n* lines will be a sequence of comma-separated integers that stands for one row of the matrix.
Your task is to output the type of matrix and should be one of these strings: `diagonal`, `scalar`, `identity`, `non-diagonal`. The type you output should be the most appropriate one for the given matrix.

```
n=int(input())
mat=[]
a=True
b=True
c=True
for i in range(n):
    t=[]
    t=input().split(',')
    for j in range(n):
        t[j]=int(t[j])
mat.append(t)
for i in range(n):
    for j in range(n):
        if i!=j:
            if mat[i][j]!=0:
                a=False
        if mat[i][i]!=mat[j][j]:
            b=False
        if mat[i][i]!=1:
            c=False
if a and b and c:
    print('identity')
```

```
elif a and b:
    print('scalar')
elif a:
    print('diagonal')
else:
    print('non-diagonal')
```

ProgQuiz-M4

There are five boxes arranged from left to right. You keep adding a variable number of coins sequentially in each box. Start from box-1 and keep going right. Once you reach the last box, head back to box-1 and then keep adding coins. In any given turn, the number of coins added to a box is always less than 10.
Find the box which has the maximum number of coins. If there are two boxes which have the same maximum number of coins, output the smaller of the two box numbers. The sequence of coins is represented by a string. For example, if the input is `3972894910`, this is how coins are added:

Box	Coins
1	3 + 9 = 12
2	9 + 4 = 13
3	7 + 9 = 16
4	2 + 1 = 3
5	8 + 0 = 8

In this case, 3is the output as box-3 has the maximum number of coins in it.

```
l = [int(i) for i in input()]
x = [0,0,0,0,0]
k=0
for i in l:
    if k<5:
        x[k]+=i
    else:
        k=0
        x[k]+=i
    k+=1
print(x.index(max(x))+1)
```

ProgQuiz-M5

Accept a positive integer A as input and print the dimensions of all rectangles with integer sides that have an area equal to A. Treat a square as a rectangle with equal sides. In each line, print the dimension of one rectangle as a pair of comma separated integers — `a,b` — such that $a \leq b$. The rectangle with sides a and b should be printed exactly once. That is, (a,b) and (b,a) represent the same rectangle.

```
a=int(input())
for i in range(1,a+1):
    if a%i==0:
        if i<=(a/i):
            print(i,int(a/i),sep=(','))
```

ProgQuiz-M6

Sort a list `L` of items in non-decreasing order and store it in the list `sorted_L`. All items in the list are of the same type. This common type could be `int`, `float` or `str`. The list `L` is already given to you.

---

(1) You must write your solution within the function. Indent all your code by four spaces.

(2) You don't need to accept the input or print the output to the console.

(3) You are not allowed the use of built-in sort functions.

```
def sort(L):
    # Enter your code below this line
    # Indent all your code by four spaces
    for i in range(len(L)):
      for j in range(i,len(L)):
        if L[i]>=L[j]:
          L[i],L[j]=L[j],L[i]
sorted_L=L
    # Enter your code above this line
    # Indent all your code by four spaces
    return sorted_L
```

PQuiz-1

Accept a positive integer n as input and find the print the smallest integer that is divisible by all the integers in the range $[1,n]$, endpoints inclusive.

```
n=int(input())
c=1
while 1:
    flag=True
    for i in range(1,n+1):
        if c%i==0 and c!=i:
            continue
        else:
            flag=False
            break
    if flag:
        print(c)
        break
    c=c+1
```

PQuiz-2

Consider a sequence of words. A sub-sequence is a subset of consecutive words in this sequence. For example, given the following sequence:

```
one,two,order,real,long,tight,tree,cool,lot,trouble
```

The following are some sub-sequences:

(1) one,two,order

(2) real,long,tight,tree

(3) cool

(4) one,two,order,real,long,tight,tree,cool,lot,trouble

Note that `one,lot` does not form a sub-sequence as far as this problem is concerned. (3) and (4) are valid sub-sequences even though they are quite trivial in nature.

A sub-sequence is said to have the antakshari property if the last letter of every word in the sub-sequence is equal to the first letter in the next word. For example, in the above sequence, we have the following sub-sequences with this property:

```
cool,lot
cool,lot,trouble
two,order,real
two,order,real,long
```

Your task is to find the length of the longest sub-sequence with the antakshari property. In the above sequence, the longest sub-sequence with this property has length 4.

Accept a sequence of comma separated words as input and print the length of the longest sub-sequence with the antakshari property. All words in the sequence will be in lower case.

```
l=input().split(',')
c=1
maxi=1
for i in range(len(l)):
    if i!=(len(l)-1):
        if l[i][-1]==l[i+1][0]:
            c=c+1
            if c>maxi:
                maxi=c
        else:
            if c>maxi:
                maxi=c
                c=1
            else:
                c=1
print(maxi)
```

PQuiz-3

This problem is about reversing a square matrix along row or column.

Reversing a matrix along the rows is to perform the following operation:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \xrightarrow{row} \begin{bmatrix} a_{22} & a_{21} & a_{22} \\ a_{10} & a_{11} & a_{12} \\ a_{00} & a_{01} & a_{02} \end{bmatrix}$$

Reversing a matrix along the columns is to perform the following operation:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \xrightarrow{column} \begin{bmatrix} a_{02} & a_{01} & a_{00} \\ a_{12} & a_{11} & a_{10} \\ a_{22} & a_{21} & a_{00} \end{bmatrix}$$

The first line of the input will be an integer *n*, which denotes the dimension of the square matrix. Each of the next *n* lines in the input will have a sequence of *n* comma-separated integers. The last line in the input will be one of these two words: `row` or `column`. If it is row, then reverse the matrix along the row, else, reverse it along the column.
Print the reversed matrix as output: each line should contain one row of the matrix as a sequence of comma-separated integers.

```
n=int(input())
mat=[]
for i in range(n):
    a=input().split(',')
    for j in range(n):
        a[j]=int(a[j])
mat.append(a)
s=input()
final_mat=[]
if s=='row':
    for i in range(n-1,-1,-1):
        a=[]
        for j in range(n):
a.append(mat[i][j])
final_mat.append(a)
if s=='column':
    for i in range(n):
        a=[]
        for j in range(n-1,-1,-1):
a.append(mat[i][j])
final_mat.append(a)
for i in range(n):
    for j in range(n):
        if j!=n-1:
            print(final_mat[i][j],end=',')
        else:
            print(final_mat[i][j])
```

PQuiz-4

A string `str_1` is a substring of another string `str_2`, if `str_1` is present as a sequence of consecutive characters in `str_2`. For example, `got` is a substring of `gottingen`, whereas `got` is *not* a substring of `goat`.

Accept a sequence of comma separated words as input. Print that word in the sequence which is a substring of every other word in the sequence.

If you do not find any word that is a common substring of all words in the sequence, print `None`. Assume that all the words will be in lower case.

```
def subset(s,word):
    for i in s:
        if word not in i:
            return False
    return True

s=input().split(',')
flag=True
for i in range(len(s)):
    if subset(s,s[i]):
        print(s[i])
        flag=False
        break
if flag:
    print('None')
```

PQuiz-5

A number is called a **double palindrome** if both the number and its square are palindromes. For example, 11 is double palindrome as both 11 and 121 are palindromes. Accept a positive integer n as input and print all the double palindromes less than or equal to n in ascending order.

```
n=input()
l=[]
a=1
while a<=int(n):
  p1=False
  r=''
  k=''
  j=a**2
  a=str(a)
  j=str(j)
  m=len(a)
  z=len(j)
  for i in range(m-1,-1,-1):
    r+=a[i]
  if a==r:
    p1=True
  if p1:
    for i in range(z-1,-1,-1):
      k+=j[i]
    if j==k:
l.append(a)
  a=int(a)
  a+=1
for i in range(len(l)):
  print(l[i])
```

PQuiz-6

Three rectangular matrices A, B and C are provided to you. You need to compute the product of these three matrices: $A \times B \times C$. Store the results of this matrix multiplication in a matrix named as `prod`. Each of these matrices is a list of lists.

You do not have to accept input from the console or print the output to the console. You just have to write your code within the function provided. Make sure to indent all your code by four spaces.

```
def multiply(A, B, C):
    # Write your code below this line
    # Indent all your code by four spaces so that it aligns with this comment
    mat1=[]
    for i in range(len(A)):
        t=[]
        for j in range(len(B[0])):
            temp=0
for k in range(len(A[0])):
                temp+=A[i][k]*B[k][j]
t.append(temp)
        mat1.append(t)
    prod=[]
    for i in range(len(mat1)):
        t=[]
        for j in range(len(C[0])):
            temp=0
for k in range(len(mat1[0])):
                temp+=mat1[i][k]*C[k][j]
t.append(temp)
prod.append(t)
    # Write your code above this line
    # Indent all your code by four spaces so that it aligns with this comment
    return prod
```

ProgQuiz-2-M1

`para` is a sequence of space-separated words. All words will be in lower case. There will be a single space between consecutive words. The string has no other special characters other than the space.

Write a function named `exact_count` that accepts the string `para` and a positive integer n*n* as arguments. You have to return `True` if there is at least one word in `para` that occurs exactly n*n* times, and `False` otherwise.

```
defexact_count(para,n):
'''
    Arguments:
      para: string
      n: integer
    Return:
      result: bool
  '''

```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

```
def exact_count(para, n):
    para=para.split(' ')
    d={}
    for i in para:
        if i not in d:
            d[i]=1
        else:
            d[i]=d[i]+1
    l=d.values()
    for i in l:
        if i==n:
            return True
    return False
```

ProgQuiz-2-M2

Your task is to do simple word problems such as this:

```
one plus two plus three
```
The answer is 6.

Accept a sequence of space-separated words as input. Each word is either a digit from "zero" to "nine" (endpoints inclusive) or one of the two operands: "plus" or "minus". The operands and operators alternate in the sequence. In other words, no two consecutive words will be of the same type.

You have to find the solution of this arithmetic problem and print the answer as an integer. Evaluate the expression without introducing brackets anywhere. That is,

```
minus one plus two minus three
```
is just $-1+2-3$.

```
l=input().split(' ')
sum=0
c=1
d={'zero':0,'one':1,'two':2,'three':3,'four':4,'five':5,'six':6,'seven':7,'eig
ht':8,'nine':9}
for i in l:
    if i=='minus':
        c=-1
        continue
    if i=='plus':
        c=1
        continue
    sum=sum+(c*d[i])
print(sum)
```

ProgQuiz-2-M3

The price of a steel rod is generally a simple function of its length. However, requirements of companies also influence the price. If you are selling rods to a company that has a preference for short rods and doesn't use too many long rods, the price distribution could look like this:

Rod-length	Price
1	10
2	20
3	20
4	5
5	3

If you have a rod of length 5 meters, you would make a lot more money by cutting the rod and selling it as two rods — one of length 3 meters and another of length 2 meters — than selling a single rod of length 5 meters.

Your task is to accept the length of a rod and the price distribution as inputs. You are allowed to make at most one cut of the rod. Find the maximum revenue that you can obtain. Assume that you can only sell rods of integer lengths.

First line of input is the length of the rod, L. The second line is a sequence of L comma separated integers that corresponds to the selling prices of rods of lengths $(1,2,3,\cdots,L-1,L)$. Print the maximum revenue that can be obtained with at most one cut of the given rod.

```
n=int(input())
l=input().split(',')
p=[]
max=0
```

```
for i in range(len(l)):
    for j in range(len(l)):
        s=0
        if i+j==(n-2):
            s=int(l[i])+int(l[j])
p.append(s)
p.append(int(l[-1]))
for i in range(len(p)):
    if p[i]>max:
        max=p[i]
print(max)
```

ProgQuiz-2-M4

Write a recursive function named `subsets` that accepts a non-empty list of distinct integers `L` as argument. It should return the list of all subsets of `L`.
(1) Each subset is to be represented as a list of numbers.

(2) The order in which you arrange the elements within a subset doesn't matter. For L = [1, 2, 3], [1, 3] and [3, 1] represent the same subset.
(3) The order in which you append the subsets to the returned list doesn't matter.

(4) The empty list is a subset for all lists.

```
defsubsets(L):
'''
    Argument:
        L: list of integers
    Return:
        result: list of lists
    '''
```

You do not have to accept input from the console or print output to the console.

```
def subsets(L):
    if len(L) == 1:
        return [[],L]
    else:
        l = []
        for sub in subsets(L[0:-1]):
l.append(sub)
l.append([L[-1]])
l.append(sub+[L[-1]])
    t=[]
    for i in l:
        if i not in t:
t.append(i)
```

```
    return t
```

ProgQuiz-2-M5

Consider an irrational number in the following form:

$$a + b\sqrt{p}$$

a,b and p are integers. Additionally, p is a prime. For all $n \geq 1$, it is known that there is a unique tuple of integer (x,y,p) such that:

$$\left(a + b\sqrt{p}\right)^n = x + y\sqrt{p}$$

For example:

$$\left(2 + 3\sqrt{5}\right)^2 = 49 + 12\sqrt{5}$$

Write a function named `compute` that accepts the integers a, b, p and n as arguments and returns a tuple of integers (x,y).

```
defcompute(a,b,p,n):
'''
    Arguments:
        a: integer
        b: integer
        p: prime
        n: positive integer
    Return:
        result: (x, y): tuple of integers
  '''
```

You do not have to accept input from the user or print output to the console. You just have to write the function definition.

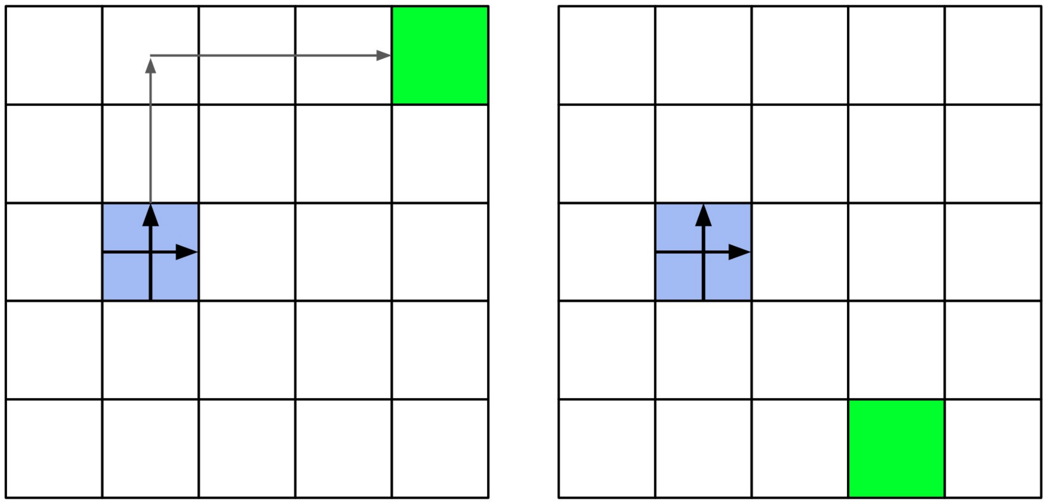
```
def compute(a, b, p, n):
x,y=a,b
    for i in range(n-1):
x,y=((a*x)+(b*y*p)),((b*x)+(a*y))
    return ((x,y))
```

ProgQuiz-2-M6

Consider a grid-world that is inhabited by an ant (BLUE). The ant can move only in two directions: UP or RIGHT. The ant has sensed the presence of a source of food somewhere in the grid. Your task is twofold:

- Determine if the ant can reach the food source.
- If it can, find out the number of steps it has to take.

For example, in the grid-world on the left, the ant can reach the food source in five steps. On the right, it can't.



The grid-world is represented as a matrix of strings: 'B' stands for the initial position of the ant, 'W' stands for an empty cell and 'G' stands for the food source. For example, the grid-world on the left is represented as:

$$\begin{bmatrix} W & W & W & W & G \\ W & W & W & W & W \\ W & B & W & W & W \\ W & W & W & W & W \\ W & W & W & W & W \end{bmatrix}$$

Write a function named `is_reachable` that accepts a n \times n$n \times n$ matrix of strings named `grid` as argument. Return `(True, steps)` if the ant can reach the food source, where `steps` is the number of steps the ant needs to take. If it can't reach the food source, return `(False, None)`

```
defis_reachable(grid):
'''
    Argument:
      grid: matrix of strings (upper-case characters)
    Return:
      result: tuple, either (True, int) or (False, None)
  '''
```

You do not have to accept the input from the user or print output to the console.

```
def is_reachable(grid):
    n=len(grid)
    for i in range(n):
        for j in range(n):
            if grid[i][j]=='B':
ib=i
jb=j
            if grid[i][j]=='G':
ig=i
jg=j
    if ib>=ig and jb<=jg:
        a=(ib-ig)+(jg-jb)
        return ((True,a))
else :
        return ((False,None))
```