**https://www.hackerrank.com/domains/python?filters%5Bstatus%5D%5B%5D=unsolved&filters%5Bstatus%5D%5B%5D=solved&filters%5Bsubdomains%5D%5B%5D=py-basic-data-types&badge\_type=python**

**Python If-Else**

**import** math

**import** os

**import** random

**import** re

**import** sys

n = **int**(**input**(""))

**if** 1<=n<=100:

**if** n%2 == 1 **and** 1<=n<=100 :

**print**("Weird")

**elif** n%2 == 0 **and** n<5 :

**print**("Not Weird")

**elif** n%2 == 0 **and** 6<=n<=20:

**print**("Weird")

**elif** n%2 == 0 **and** 20<=n<=100:

**print**("Not Weird")

**else**:

**print**("")

# Arithmetic Operators

a = **int**(**input**(""))

b = **int**(**input**(""))

**print**(a + b)

**print**(a - b)

**print**(a \* b)

# Python: Division

a = **int**(**input**(""))

b = **int**(**input**(""))

**print**(a//b)

**print**(a/b)

# Loops

n = **int**(**input**(""))

**for** i **in** **range**(n):

**print**(i \* i)

# Write a function

**def** is\_leap(year):

**if** year % 400 == 0:

**return** **True**

**if** year % 100 == 0:

**return** **False**

**if** year % 4 == 0:

**return** **True**

**else**:

**return** **False**

year = **int**(**input**())

**print**(is\_leap(year))

# Print Function

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

end=”” ---- used to print without spaces

LIST COMPREHENSION

n = 2

print([[i,j] for i in range(n) for j in range(2)])

**[0, 1] [0, 1]**

**It will print [[0,0],[0,1],[1,0],[1,1]]**

x = **int**(**input**())

y = **int**(**input**())

z = **int**(**input**())

n = **int**(**input**())

**print**([[i,j,k] **for** i **in** **range**(x+1) **for** j **in** **range**(y+1) **for** k **in** **range**(z+1) **if** (i+j+k) != n ])

**input:**

**1**

**1**

**1**

**2**

**o/p: [[0, 0, 0], [0, 0, 1], [0, 1, 0], [1, 0, 0], [1, 1, 1]]**

**\*\*\*\*\*\*\*\*\*\***

n = int(input())  
list = []  
for i in range(n):  
 list.insert(i, int(input()))  
  
print(list)  
list.sort()  
print(list)  
print(list[-2])

**\*\*\*\*\*\*\*\*\*\*\***

# Find the Runner-Up Score!

n = int(input())  
array = list(map(int, input().split()))  
array = sorted(array)  
print(array)  
print(array[-2])

# Nested Lists

students = []  
sec\_name = []  
second\_low = 0  
n = int(input("Input number of students: "))  
for \_ in range(n):  
 s\_name = input("Name: ")  
 score = float(input("Grade: "))  
 students.append([s\_name,score])  
print("\nNames and Grades of all students:")  
print(students)  
order =sorted(students, key = lambda x: int(x[1]))  
for i in range(n):  
 if order[i][1] != order[0][1]:  
 second\_low = order[i][1]  
 break  
print("\nSecond lowest grade: ",second\_low)  
sec\_student\_name = [x[0] for x in order if x[1] == second\_low]  
sec\_student\_name.sort()  
print("\nNames:")  
for s\_name in sec\_student\_name:  
 print(s\_name)

# String Split and Join

**def** split\_and\_join(line):

    line  =  line.split(" ")

    line =  "-".join(line)

**return** line

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

    line = **input**()

    result = split\_and\_join(line)

**print**(result)

# What's Your Name?

def print\_full\_name(a, b):  
 print("Hello " + a + " " + b + "! You just delved into python.")   
  
if \_\_name\_\_ == '\_\_main\_\_':  
 first\_name = input()  
 last\_name = input()  
 print\_full\_name(first\_name, last\_name)

# Find a string

**def** count\_substring(string, sub\_string):

    count = 0

**for** i **in** **range**(**len**(string) - **len**(sub\_string) + 1):

**if** string[i:i + **len**(sub\_string)] == sub\_string:

            count += 1

**return** count

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

    string = **input**().strip()

    sub\_string = **input**().strip()

    count = count\_substring(string, sub\_string)

**print**(count)

# sWAP cASE

**def** swap\_case(s):

    result =s.swapcase()

**return** result

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

    s = **input**()

    result = swap\_case(s)

**print**(result)

# Capitalize!

**def** solve(s):

    a = s.split()

**for** i **in** a :

        s =  s.replace(i, i.capitalize())

**return** s

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

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

s = **input**()

result = solve(s)

fptr.write(result + '\n')

    fptr.close()

# basic data types: Tuples 27-09-2020

n = **int**(**input**())

t = **tuple**(**map**(**int**, **input**().split()))

**print**(**hash**(t))

# math: Integers Come In All Sizes

a = **int**(**input**())

b = **int**(**input**())

c = **int**(**input**())

d = **int**(**input**())

**print**(a\*\*b  + c\*\*d)

# strings: Text Wrap

**import** textwrap

**def** wrap(string, max\_width):

    result = textwrap.fill(string,max\_width)

# result = textwrap.wrap(string,max\_width)

**return** result

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

    string, max\_width = **input**(), **int**(**input**())

    result = wrap(string, max\_width)

**print**(result)

Your Output (stdout) comment: input : abcdefghy,3

o/p: [‘abc’ , ‘def’ , ‘ghy’]

* **ABCD**
* **EFGH**
* **IJKL**
* **IMNO**
* **QRST**
* **UVWX**
* **YZ**

# Math: Mod Divmod

a = **int**(**input**())

b = **int**(**input**())

**print**(a//b)

**print**(a%b)

**print**(**divmod**(a,b))

# math :Power - Mod Power

**from** math **import** \*

a = **int**(**input**())

b = **int**(**input**())

m = **int**(**input**())

c = a \*\* b

**print**(c)

d = c % m

**print**(d)

**basic data types:Lists 29-09-2020**

N = int(input())  
list = []  
for i in range(N):  
 condition = str(input())  
 if "insert" in condition:  
 position = int(condition[7])  
 value = int(condition[9:])  
 list.insert(position,value)  
 elif "print" in condition:  
 print(list)  
 elif "remove" in condition:  
 list.remove(int(condition[7:]))  
 elif "append" in condition:  
 list.append(int(condition[7:]))  
 elif "sort" in condition:  
 list.sort()  
 elif "pop" in condition:  
 list.pop()  
 else:  
 list.reverse()

**Basic datatypes: Finding the percentage**

n = int(input())  
student\_marks = {}  
for \_ in range(n):  
 name, \*line = input().split()  
 scores = list(map(float, line))  
 student\_marks[name] = scores  
  
query\_name = input()  
x = (sum(student\_marks[query\_name])/len(student\_marks[query\_name]))  
print("{:.2f}".format(x))

print a value in two decimal places

a\_float = 3.14159

formatted\_float = "{:.2f}".format(a\_float)

**strings: Mutations**

def mutate\_string(string, position, character):  
 l = list(string)  
 l[position] = character  
 string = ''.join(l)  
 s\_new = string  
 return s\_new  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 s = input()  
 i, c = input().split()  
 s\_new = mutate\_string(s, int(i), c)  
 print(s\_new)

**Introduction to Sets 1-10-2020**

**def** average(array):

    s = **set**(arr)

    result = **sum**(s)/**len**(s)

**return** result

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

    n = **int**(**input**())

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

    result = average(arr)

**print**(result)

**Set .discard(), .remove() & .pop()**

n = **int**(**input**())

s = **set**(**map**(**int**, **input**().split()))

N = **int**(**input**())

**for** i **in** **range**(N):

    condition = **str**(**input**())

**if** "remove" **in** condition:

        position = **int**(condition[7])

        s.remove(position)

**elif** "discard" **in** condition:

        position = **int**(condition[8])

        s.discard(position)

**elif** "pop" **in** condition:

        s.pop()

**print** (**sum**(s))

**Set .union() Operation**

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

n = **int**(**input**())

arr1 = **list**(**map**(**int**, **input**().split()))

s1 = **set**(arr1)

b = **int**(**input**())

arr2 = **list**(**map**(**int**, **input**().split()))

s2 = **set**(arr2)

k = s2.union(s1)

l = s1.union(s2)

m = k.union(l)

**print**(**len**(m))

**Set .add()**

N = **int**(**input**())

**list** = []

**for** \_ **in** **range**(N):

    country\_name = **input**()

**list**.append(country\_name)

s = **set**(**list**)

**print**(**len**(s))

**Set .intersection() Operation**

n = **int**(**input**())

arr1 = **list**(**map**(**int**,**input**().split()))

s1 = **set**(arr1)

b = **int**(**input**())

arr2 = **list**(**map**(**int**,**input**().split()))

s2 = **set**(arr2)

k = s1.intersection(s2)

**print**(**len**(k))

**Symmetric Difference**

a = **int**(**input**())

M = **set**(**map**(**int**, **input**().split()))

b = **int**(**input**())

N = **set**(**map**(**int**,**input**().split()))

diff1 = M-N

diff2 = N-M

j = diff1.union(diff2)

k = diff2.union(diff1)

l = j.union(k)

**for** \_ **in** **range**(**len**(l)):

**print**(**min**(l))

    l.remove(**min**(l))

**Set .difference() Operation**

E = **int**(**input**())

English\_newspaper = **set**(**map**(**int**, **input**().split()))

F = **int**(**input**())

French\_newspaper = **set**(**map**(**int**, **input**().split()))

only\_English  = English\_newspaper - French\_newspaper

**print**(**len**(only\_English))

**Set .symmetric\_difference() Operation**

E = **int**(**input**())

English\_newspapers = **set**(**map**(**int**, **input**().split()))

F = **int**(**input**())

French\_newspapers = **set**(**map**(**int**, **input**().split()))

j = English\_newspapers & French\_newspapers

k = English\_newspapers - French\_newspapers

l = French\_newspapers - English\_newspapers

m = k.union(l)

o = m - j

**print**(**len**(o))

**itertools.product() : 04-10-2020**

**from** itertools **import** product

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

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

**print**(\*(product(A,B)))

**itertools.combinations()**

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

S, K=  **input**().split()

C = [ ''.join(c) **for** i **in** **range**(1,**int**(K)+1) **for** c **in** combinations(**sorted**(S),i) ]

**print**(\*C, sep='\n')

**find a string:**

**def** count\_substring(string, sub\_string):

    count = 0

**for** i **in** **range**(**len**(string) - **len**(sub\_string)+ 1):

**if** string[i:i + **len**(sub\_string)] == sub\_string:

            count +=1

**return** count

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

    string = **input**().strip()

    sub\_string = **input**().strip()

    count = count\_substring(string, sub\_string)

**print**(count)

**nested lists**

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

**list** = []

    q = []

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

        name = **input**()

        score = **float**(**input**())

        x= [name,score]

        q.append(score)

**list**.append(x)

q.sort()

w = q[1]

**if** q[1] == q[0]:

    w = q[2]

**if** q[1] == q[0] == q[2]:

    w = q[3]

**list**.sort()

**for** i **in** **range**(**len**(**list**)):

**if** **list**[i][1] == w:

**print**(**list**[i][0])

**String Formatting 08-10-2020**

**def** print\_formatted(number):

    width = **len**("{0:b}".**format**(number))

**for** i **in** **range**(1, number + 1):

**print**("{0:{w}d} {0:{w}o} {0:{w}X} {0:{w}b}".**format**(i, w = width))

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

    n = **int**(**input**())

    print\_formatted(n)

**String Validators**

s = **input**()

contains\_alphanumeric = **any**(**map**(**str**.isalnum, s))

**print**(contains\_alphanumeric)

contains\_alphabetics = **any**(**map**(**str**.isalpha, s))

**print**(contains\_alphabetics)

contains\_digit = **any**(**map**(**str**.isdigit, s))

**print**(contains\_digit)

contains\_lowercase = **any**(**map**(**str**.islower, s))

**print**(contains\_lowercase)

contains\_uppercase = **any**(**map**(**str**.isupper, s))

**print**(contains\_uppercase)

**Text Alignment**

*#Replace all \_\_\_\_\_\_ with rjust,   or center.*

thickness = **int**(**input**()) *#This must be an odd number*

c = 'H'

*#Top Cone*

**for** i **in** **range**(thickness):

**print**((c\*i).rjust(thickness-1)+c+(c\*i).ljust(thickness-1))

*#  top Pillars*

**for** i **in** **range**(thickness+1):

**print**((c\*thickness).center(thickness\*2)+(c\*thickness).center(thickness\*6))

*#Middle Belt*

**for** i **in** **range**((thickness+1)//2):

**print**((c\*thickness\*5).center(thickness\*6))

*#Bottom Pillars*

**for** i **in** **range**(thickness+1):

**print**((c\*thickness).center(thickness\*2)+(c\*thickness).center(thickness\*6))

*#Bottom Cone*

**for** i **in** **range**(thickness):

**print**(((c\*(thickness-i-1)).rjust(thickness)+c+(c\*(thickness-i-1)).ljust(thickness)).rjust(thickness\*6))

**Check Subset**

T = **int**(**input**())

l = []

**for** i **in** **range**(T):

    N1 = **int**(**input**())

    set\_A = **set**(**map**(**int**, **input**().split()))

    N2 = **int**(**input**())

    set\_B = **set**(**map**(**int**, **input**().split()))

    x = set\_A.issubset(set\_B)

    l.append(x)

**for** values **in** l:

**print**(values)

**Set Mutations**

N1 = **int**(**input**())

set\_A = **set**(**map**(**int**, **input**().split()))

N2 = **int**(**input**())

**for** i **in** **range**(N2):

    k = **input**().split()

**if** "intersection\_update" **in** k:

        set\_B = **set**(**map**(**int**, **input**().split()))

        set\_A.intersection\_update(set\_B)

**if** "update" **in** k:

        set\_B = **set**(**map**(**int**, **input**().split()))

        set\_A.update(set\_B)

**if** "symmetric\_difference\_update" **in** k:

        set\_B = **set**(**map**(**int**, **input**().split()))

        set\_A.symmetric\_difference\_update(set\_B)

**if** "difference\_update" **in** k:

        set\_B = **set**(**map**(**int**, **input**().split()))

        set\_A.difference\_update(set\_B)

**print**(**sum**(set\_A))

**Check Strict Superset 09-10-2020**

set\_A = **set**(**map**(**int**,**input**().split()))

N = **int**(**input**())

**for** i **in** **range**(1):

    temp1 = **set**(**map**(**int**,**input**().split()))

    temp2 = **set**(**map**(**int**,**input**().split()))

**if** set\_A.issuperset(temp1) **and** set\_A.issuperset(temp2):

**print**("True")

**else**:

**print**("False")

**Exceptions 10-10-2020**

N = **int**(**input**())

**for** i **in** **range**(N):

    a,b = **input**().split()

**try**:

**print**(**int**(a) // **int**(b))

**except** ZeroDivisionError **as** e:

**print**("Error Code: integer division or modulo by zero")

**except** ValueError **as** e:

**print**("Error Code:", e)

**Incorrect Regex**

**import** re

T = **int**(**input**())

**for** i **in** **range**(T):

    S = **input**()

**try**:

**print**(**bool**(re.**compile**(S)))

**except**:

**print**("False")

**itertools.permutations()**

**from** itertools **import** \*

S, K = **input**().split()

**for** i **in** permutations(**sorted**(S),**int**(K)):

**print**("".join(i))

**itertools.combinations\_with\_replacement()**

**from** itertools **import** combinations\_with\_replacement

S, K=  **input**().split()

L = **str**(S)

**for** i **in** combinations\_with\_replacement(**sorted**(L),**int**(K)):

**print**("".join(i))

**Designer Door Mat**

N, M = **map**(**int**,**input**().split())

**for** i **in** **range**(0,N//2):

**print** (('.|.'\*i).rjust((M-2)//2,'-')+'.|.'+('.|.'\*i).ljust((M-2)//2,'-'))

**print** ('WELCOME'.center(M,'-'))

**for** i **in** **reversed**(**range**(0,N//2)):

**print** (('.|.'\*i).rjust((M-2)//2,'-')+'.|.'+('.|.'\*i).ljust((M-2)//2,'-'))

Input (stdin)

**7 21**

Expected Output

* **---------.|.---------**
* **------.|..|..|.------**
* **---.|..|..|..|..|.---**
* **-------WELCOME-------**
* **---.|..|..|..|..|.---**
* **------.|..|..|.------**
* **---------.|.---------**

**Triangle Quest 2 13-10-2020**

**for** i **in** **range**(1,**int**(**input**())+1):

**print**((((10\*\*i)-1)//9) \*\* 2)

**Triangle Quest**

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

**print**(((10\*\*i)//9) \* i)

**Find Angle MBC**

**import** math

AB = **int**(**input**())

BC = **int**(**input**())

AC1 = ((AB\*\*2) + (BC\*\*2))

AC = ((math.sqrt(AC1)))

MC  = (AC/2)

BC = BC/2

**print**(**str**(**int**(**round**(math.degrees(math.acos(BC/MC)))))+'°')

**The Captain's Room 14-10-2020**

seen = **set**()

K = **int**(**input**())

List1 = **list**(**map**(**int**,**input**().split()))

set1 = **set**()

set2 = **set**()

**for** i **in** List1:

**if** i **in** set1:

        set2.add(i)

**else**:

        set1.add(i)

set3 = set1.difference(set2)

**print**(**list**(set3)[0])

**Polar Coordinates**

**import** cmath

z = **complex**(**input**())

**print**(\*cmath.polar(z), sep='\n')

o/p

* **2.23606797749979**
* **1.1071487177940904**

here ‘\*’ means unpack the values

with out ‘\*’ output is

(6.08276253029822, 1.4056476493802699)

**Calendar Module**

**import** calendar

dd,mm,yyyy = **input**().split()

**print** (calendar.day\_name[calendar.weekday(**int**(yyyy),**int**(dd),**int**(mm))].upper())

**Arrays : numpy 15-10-2020**

**import** numpy

arr2 = []

**def** arrays(arr):

**for** i **in** **range**(1,**len**(arr)+1):

        x = **float**(arr[**len**(arr)-i])

        arr2.append(x)

    result = numpy.array(arr2)

**return** result

arr = **input**().strip().split(' ')

result = arrays(arr)

**print**(result)

# Min and Max : numpy

**import** numpy

N, M = **map**(**int**, **input**().split())

x = numpy.array([**input**().split() **for** \_ **in** **range**(N)],**int**)

**print**(numpy.**max**(numpy.**min**(x, axis=1), axis=0))

**Standardize Mobile Number Using Decorators**

**def** wrapper(f):

**def** fun(l):

        f("+91 "+ c[-10: -5] + " " + c[-5:] **for** c **in** l)

*# complete the function*

**return** fun

@wrapper

**def** sort\_phone(l):

**print**(\***sorted**(l), sep='\n')

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

    l = [**input**() **for** \_ **in** **range**(**int**(**input**()))]

    sort\_phone(l)

**Built-ins : Input()**

x,k =**map**(**int**,**input**().split())

l = **eval**(**input**())

**print**(k==l)

# Built-ins : Python Evaluation

**eval**(**input**())

**Built-ins : Zipped!**

N, X = **map**(**int**, **input**().split())

l1 = []

**for** i **in** **range**(X):

    k = **map**(**float**, **input**().split())

    l1.append(k)

**for** j **in** **zip**(\*l1):

**print**( **sum**(j)/**len**(j) )

17 – 10 - 2020

**python functional : Map and Lambda Function**

cube = **lambda** x: x \* x \* x  *# complete the lambda function*

**def** fibonacci(n):

    num1 = 0

    num2 = 1

    count = 2

**list** = []

**if** n == 1:

**list** = [0]

**elif** n == 2 :

**list** = [0,1]

**elif** 2 < n <=15:

**while** count < n <= 15:

**if** **list** == []:

**list**.append(num1)

**list**.append(num2)

**else**:

                num3 = num1 + num2

**list**.append(num3)

                temp = num3

                num1 = num2

                num2 = temp

                count += 1

**return** **list**

*# return a list of fibonacci numbers*

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

    n = **int**(**input**())

**print**(**list**(**map**(cube, fibonacci(n))))

**Built-ins : Any or All**

n = **int**(**input**())

l = **input**().split()

**print**(**all**(**int**(i)>0 **for** i **in** l) **and** **any**(j == j[::-1] **for** j **in** l))

18-10-2020

Collections: collections.counter()

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

X = **int**(**input**())

shoe\_sizes = Counter(**map**(**int**, **input**().split()))

N = **int**(**input**())

total\_income = 0

**for** i **in** **range**(N):

    size, price = **map**(**int**, **input**().split())

**if** shoe\_sizes[size]:

        total\_income += price

        shoe\_sizes[size] -= 1

**print**(total\_income)

**19-10-2020**

**Collections.namedtuple()**

**from** collections **import** namedtuple

N = **int**(**input**())

column\_names = **input**()

total = 0

**for** i **in** **range**(N):

    students = namedtuple('student',column\_names)

    row1,row2,row3,row4 = **input**().split()

    student = students(row1,row2,row3,row4)

    total += **int**(student.MARKS)

**print**((total/N))

**2/6 test cases failed**

**Collections.deque()**

**from** collections **import** deque

N = **int**(**input**())

d = deque()

**for** i **in** **range**(N):

    condition = **input**().split()

**if**  'append' **in** condition[:-1]:

        d.append(condition[-1])

**if** 'appendleft' **in**  condition[:-1]:

        d.appendleft(condition[-1])

**if** "pop" **in** condition:

        d.pop()

**if** "popleft" **in** condition:

        d.popleft()

**print**(\*d)

21-10-2020

**Collections.OrderedDict() :**

**from** collections **import** OrderedDict

N = **int**(**input**())

OD = OrderedDict()

**for** \_ **in** **range**(N):

    item = **input**().split(" ")

    price = **int**(item[-1])

    item\_name = " ".join(item[:-1])

**if** OD.get(item\_name) :

        OD[item\_name] += price

**else**:

        OD[item\_name] = price

**for** item\_name,total\_price **in** OD.items():

**print**(item\_name,total\_price)

**Collections.deque() :**

**from** collections **import** deque

N = **int**(**input**())

d = deque()

**for** i **in** **range**(N):

    condition = **input**().split()

**if**  'append' **in** condition[:-1]:

        d.append(condition[-1])

**if** 'appendleft' **in**  condition[:-1]:

        d.appendleft(condition[-1])

**if** "pop" **in** condition:

        d.pop()

**if** "popleft" **in** condition:

        d.popleft()

**print**(\*d)

**DefaultDict Tutorial 22-10-2020**

**from** collections **import** defaultdict

DD = defaultdict(**list**)

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

**for** i **in** **range**(1,n+1):

    DD[**input**()].append(**str**(i))

**for** i **in** **range**(m):

**print** (' '.join(DD[**input**()]) **or** -1 )

**23-10-2020**

**classes : Find the Torsional Angle**

**import** math

**class** Points(**object**):

**def** \_\_init\_\_(**self**, x, y, z):

**self**.x = **float**(x)

**self**.y = **float**(y)

**self**.z = **float**(z)

**def** \_\_sub\_\_(**self**, no):

**return** Points((**self**.x - no.x),(**self**.y - no.y),(**self**.z - no.z))

**def** dot(**self**, no):

**return** (**self**.x \* no.x+**self**.y \* no.y+**self**.z \* no.z)

**def** cross(**self**, no):

**return** Points((**self**.y \* no.z - **self**.z \* no.y),(**self**.z \* no.x - **self**.x \* no.z),(**self**.x \* no.y - **self**.y \* no.x))

**def** absolute(**self**):

**return** **pow**((**self**.x \*\* 2 + **self**.y \*\* 2 + **self**.z \*\* 2), 0.5)

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

    points = **list**()

**for** i **in** **range**(4):

        a = **list**(**map**(**float**, **input**().split()))

        points.append(a)

    a, b, c, d = Points(\*points[0]), Points(\*points[1]), Points(\*points[2]), Points(\*points[3])

    x = (b - a).cross(c - b)

    y = (c - b).cross(d - c)

    angle = math.acos(x.dot(y) / (x.absolute() \* y.absolute()))

**print**("%.2f" % math.degrees(angle))

**Regex and parsing : Detect Floating Point Number**

**import** re

x = **int**(**input**())

**for** i **in** **range**(x):

**print** (**bool**(re.match(r'^[-+]?[0-9]\*\.[0-9]+$', **input**())))

**strings : Merge the Tools! 24-10-2020**

**def** merge\_the\_tools(string, k):

**from** collections **import** OrderedDict

**for** x **in** **range**(0,**len**(string),k):

**print**(''.join(**list**(OrderedDict.fromkeys(string[x:x+k]))))

*# your code goes here*

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

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

    merge\_the\_tools(string, k)

# Numpy : Concatenate 09-11-2020

**import** numpy

N,M,P = **map**(**int**,**input**().split())

x = numpy.array([**input**().split() **for** i **in** **range**(N)],**int**)

y = numpy.array([**input**().split() **for** i **in** **range**(M)],**int**)

**print**(numpy.concatenate((x,y),axis = 0))

# Numpy : Dot and Cross

**import** numpy

n = **int**(**input**())

A = numpy.array([**input**().split() **for** i **in** **range**(n)],**int**)

B = numpy.array([**input**().split() **for** i **in** **range**(n)], **int**)

**print**(numpy.dot(A,B))

# Numpy : Sum and Prod 10-11-2020

**import** numpy

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

arr1 = numpy.array([**input**().split() **for** i **in** **range**(n)],**int**)

x = numpy.**sum**(arr1,axis = 0)

**print**(numpy.prod(x,axis = 0))

# Numpy : Zeros and Ones

**import** numpy

numbers = **list**(**map**(**int**,**input**().split()))

**print**(numpy.zeros(numbers,dtype = numpy.**int**))

**print**(numpy.ones(numbers,dtype = numpy.**int**))

**Compress the String! 11-11-2020**

**from** itertools **import** groupby

**for** i,h **in** groupby(**input**()):

**print**("(" + **str**(**len**(**list**(h))) + ", " + i + ")",end = " ")

# Numpy: Shape and Reshape

**import** numpy

arr1 = numpy.array(**list**(**map**(**int**,**input**().split())))

**print** (numpy.reshape(arr1,(3,3)))

# Numpy : Array Mathematics

**import** numpy

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

a = numpy.array([**input**().split() **for** i **in** **range**(n)],**int**)

b = numpy.array([**input**().split() **for** i **in** **range**(n)], **int**)

**print** (numpy.add(a, b))

**print** (numpy.subtract(a, b))

**print** (numpy.multiply(a, b))

**print** (a//b)

**print** (numpy.mod(a, b))

**print** (numpy.power(a, b))

# Numpy : Floor, Ceil and Rint

**import** numpy

numpy.set\_printoptions(sign=' ')

A = numpy.array(**input**().split(),**float**)

**print**(numpy.floor(A))

**print**(numpy.ceil(A))

**print**(numpy.rint(A))

# collections : Company Logo 12-11-2020

**import** math

**import** os

**import** random

**import** re

**import** sys

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

[**print**(\*c) **for** c **in** Counter(**sorted**(**input**())).most\_common(3)]

# Numpy : Polynomials 13-11-2020

**import** numpy

arr = numpy.array([i **for** i **in** **input**().split()],**float**)

x = **int**(**input**())

**print**(numpy.polyval(arr,x))

# numpy : Inner and Outer

**import** numpy

A = numpy.array([c **for** c **in** **input**().split()],**int**)

B = numpy.array([c **for** c **in** **input**().split()],**int**)

**print**(numpy.inner(A,B))

**print**(numpy.outer(A,B))

# numpy : Transpose and Flatten

**import** numpy

M,N = **input**().split()

arr1 = numpy.array([**input**().split() **for** s **in** **range**(**int**(M))],**int**)

**print**(arr1.transpose())

**print**(arr1.flatten())

# numpy : Linear Algebra

**import** numpy

N = **int**(**input**())

A = numpy.array([**input**().split() **for** \_ **in** **range**(N)],**float**)

numpy.set\_printoptions(legacy='1.13')

**print** (numpy.linalg.det(A))

# numpy : Mean, Var, and Std

**import** numpy

# numpy.set\_printoptions(legacy = '1.13')

N,M = **input**().split()

A = numpy.array([**input**().split() **for** \_ **in** **range**(**int**(N))],**float**)

**print** (numpy.mean(A, axis = 1))

**print** (numpy.var(A, axis = 0))

**print**(( "%.11f" % numpy.std(A, axis = **None**)))

# 1/3 test cases failed

# Regex and parsing : Re.split()

regex\_pattern = r"[,.]" *# Do not delete 'r'.*

**import** re

**print**("\n".join(re.split(regex\_pattern, **input**())))

# 16-11-2020

# Regex and parsing : Validating UID

**import** re

no\_repeats = r"(?!.\*(.).\*\1)"

two\_or\_more\_upper = r"(?=(?:.\*[A-Z]){2,})"

three\_or\_more\_digits = r"(?=(?:.\*\d){3,})"

ten\_alphanumerics = r"[a-zA-Z0-9]{10}"

filters = no\_repeats, two\_or\_more\_upper, three\_or\_more\_digits, ten\_alphanumerics

**for** uid **in** [**input**() **for** \_ **in** **range**(**int**(**input**()))]:

**if** **all**([re.match(f, uid) **for** f **in** filters]):

**print**("Valid")

**else**:

**print**("Invalid")

# Regex and parsing : Validating phone numbers

**import** re

pattern = r"^[789]\d{9}$"

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

    A = **input**()

    x= re.match(pattern,A)

**if** x:

**print**("YES")

**else**:

**print**("NO")

# 17-11-2020

# regex & parsing: Hex color code

**import** re

n = **int**(**input**())

**for** \_ **in** **range**(n):

    K = **input**()

    matches = (re.findall(r':?.(#[0-9a-fA-F]{6}|#[0-9a-fA-F]{3})',K))

**if** matches:

**print**(\*matches,sep = '\n')

regex & parsing : **Re.start() & Re.end()**

**import** re

S = **input**()

K = **input**()

pattern = re.**compile**(K)

A = re.search(K,S)

**if** **not** A:

**print**((-1,-1))

**while** A :

**print**("({0}, {1})".**format**(A.start(),A.end()-1))

    A = pattern.search(S,A.start()+1)

**18-11-2020**

# 

# Regex & parsing : Validating and Parsing Email Addresses

**import** re

N = **int**(**input**())

**for** \_ **in** **range**(N):

    m = **input**().split()

    A = re.match(r'<[a-zA-Z][a-zA-Z0-9.\_-]+@[a-zA-Z]+\.[a-zA-Z]{1,3}>',m[1])

**if** A:

**print**(\*m)

**else**:

**pass**

# Regex & parsing : Validating Roman Numerals

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

**import** re

**print**(**str**(**bool**(re.match(regex\_pattern, **input**()))))

Regex & parsing : validating postal codes

regex\_integer\_in\_range = r"^[1-9]{1}[\d]{5}$"   *# Do not delete 'r'.*

regex\_alternating\_repetitive\_digit\_pair = r'(\d)(?=\d\1)' *# Do not delete 'r'.*

**import** re

P = **input**()

**print** (**bool**(re.match(regex\_integer\_in\_range, P))

**and** **len**(re.findall(regex\_alternating\_repetitive\_digit\_pair, P)) < 2)

**regex & parsing : Re.findall() & Re.finditer()**

**import** re

S = **input**()

pattern = (r'(?<=[QWRTYPSDFGHJKLZXCVBNM])([aeiou]{2,})([QWRTYPSDFGHJKLZXCVBNM])')

m = re.findall(pattern,S,re.I)

**if** m:

**for** i **in** **range**(**len**(m)):

**print**(m[i][0])

**else**:

**print**(-1)

**19-11-2020**

**collectins : collections.namedtuple()**

'''from collections import namedtuple

N = int(input())

column\_names = input()

total = 0

if 0 <= N <= 100:

    for i in range(N):

        students = namedtuple('student',column\_names)

        row1,row2,row3,row3 = input().split()

        student = students(row1,row2,row3,row3)

        total += int(student.MARKS)

t =((total/N))

print('{:.2f}'.format(t))

'''

**from** collections **import** namedtuple

n, Student = **int**(**input**()), namedtuple('Student', **input**())

**print**("{:.2f}".**format**(**sum**([**int**(Student(\***input**().split()).MARKS) **for** \_ **in** **range**(n)]) / n))

# regex and parsing: HTML Parser - Part 1

**from** html.parser **import** HTMLParser

**class** MyHTMLParser(HTMLParser):

**def** handle\_starttag(**self**, tag, attrs):

**print**("Start :", tag)

**for** attr **in** attrs:

**print**("->", attr[0], ">", attr[1])

**def** handle\_endtag(**self**, tag):

**print**("End   :", tag)

**def** handle\_startendtag(**self**, tag, attrs):

**print**("Empty :", tag)

**for** attr **in** attrs:

**print**("->", attr[0], ">", attr[1])

a = '\n'.join([**input**() **for** x **in** **range**(**int**(**input**()))])

parser = MyHTMLParser()

parser.feed(a)

# 20-11-2020

# regex & parsing : HTML Parser - Part 2

**from** html.parser **import** HTMLParser

**class** MyHTMLParser(HTMLParser):

**def** handle\_comment(**self**, data):

**if** "\n" **in** data:

**print** (">>> Multi-line Comment",data,sep= "\n")

**else**:

**print** (">>> Single-line Comment",data,sep = "\n")

**def** handle\_data(**self**, data):

**if** data != "\n":

**print**(">>> Data")

**print**(data)

html = ""

a = '\n'.join([**input**() **for** x **in** **range**(**int**(**input**()))])

parser = MyHTMLParser()

parser.feed(a)

parser = MyHTMLParser()

parser.feed(html)

parser.close()

# regex & parsing :

**Detect HTML Tags, Attributes and Attribute Values**

**from** html.parser **import** HTMLParser

**class** MyHTMLParser(HTMLParser):

**def** handle\_starttag(**self**, tag, attrs):

**print**(tag)

**for** attr **in** attrs:

**print**("->", attr[0], ">", attr[1])

N = **int**(**input**())

a = '\n'.join([**input**() **for** x **in** **range**(N)])

parser = MyHTMLParser()

parser.feed(a)

# regex & parsing : Regex Substitution

**import** re

N = **int**(**input**())

**for** i **in** **range**(N):

    s= **input**()

**print**(re.sub('(?<=\s)\&\&\s', 'and ', re.sub('\s\|\|\s', ' or ',s)))

# XML : XML 1 - Find the Score

**import** sys

**import** xml.etree.ElementTree **as** etree

**def** get\_attr\_number(node):

**return** **sum**(**len**(i.attrib) **for** i **in** node.**iter**())

*#return sum(len(child.attrib) for child in node.iter())*

*# your code goes here*

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

    sys.stdin.readline()

    xml = sys.stdin.read()

    tree = etree.ElementTree(etree.fromstring(xml))

    root = tree.getroot()

**print**(get\_attr\_number(root))

# 

# 21-11-2020

# Built-ins : Athlet Sort

*#!/bin/python3*

**import** math

**import** os

**import** random

**import** re

**import** sys

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

    nm = **input**().split()

    n = **int**(nm[0])

    m = **int**(nm[1])

    arr = []

**for** \_ **in** **range**(n):

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

    k = **int**(**input**())

    sorted\_arr = **sorted**(arr, key=**lambda** x: x[k])

**for** item **in** sorted\_arr:

**print**(\*item)

# Numpy : Eye and Identity

**import** numpy

numpy.set\_printoptions(legacy = '1.13')

**print**(numpy.eye(\***map**(**int**, **input**().split())))

# XML : XML2 - Find the Maximum Depth

**import** xml.etree.ElementTree **as** etree

maxdepth = 0

**def** depth(elem, level):

**global** maxdepth

**if** (level == maxdepth):

        maxdepth += 1

**for** child **in** elem:

        depth(child, level + 1)

*# your code goes here*

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

    n = **int**(**input**())

    xml = ""

**for** i **in** **range**(n):

        xml =  xml + **input**() + "\n"

    tree = etree.ElementTree(etree.fromstring(xml))

    depth(tree.getroot(), -1)

**print**(maxdepth)

# closures and Decorators: Decorators 2 - Name Directory

**import** operator

**def** person\_lister(f):

**def** inner(people):

**return** **map**(f, **sorted**(people, key=**lambda** x: **int**(x[2])))

*# complete the function*

**return** inner

@person\_lister

**def** name\_format(person):

**return** ("Mr. " **if** person[3] == "M" **else** "Ms. ") + person[0] + " " + person[1]

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

    people = [**input**().split() **for** i **in** **range**(**int**(**input**()))]

**print**(\*name\_format(people), sep='\n')

**22-11-2020**

**Debugging : Words Score**

**def** is\_vowel(letter):

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

**def** score\_words(words):

    score = 0

**for** word **in** words:

        num\_vowels = 0

**for** letter **in** word:

**if** is\_vowel(letter):

                num\_vowels += 1

**if** num\_vowels % 2 == 0:

            score += 2

**else**:

            num\_vowels % 2 == 1

            score += 1

**return** score

n = **int**(**input**())

words = **input**().split()

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

# Built-ins : ginortS(not used built – ins)

S = **input**()

d1 = ""

**for** letter **in** S:

**if** letter.islower():

        d1 +=letter

a1 = (''.join(**sorted**(d1)))

d2 = ''

**for** letter **in** S:

**if** letter.isupper():

        d2 += letter

a2 =  (''.join(**sorted**(d2)))

d3 = ''

**for** digit **in** S:

**if** digit.isdigit():

**if** **int**(digit) % 2 == 1:

            d3 += digit

a3 =  (''.join(**sorted**(d3)))

d4 = ''

**for** digit **in** S:

**if** digit.isdigit():

**if** **int**(digit) % 2 == 0:

            d4 += digit

a4 =  (''.join(**sorted**(d4)))

**print**(a1+a2+a3+a4)

# 23-11-2020

# python functional : Reduce Function

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

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

**def** product(fracs):

    t = **reduce**(**lambda** x,y : x\*y,fracs)

*# complete this line with a reduce statement*

**return** t.numerator, t.denominator

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

    fracs = []

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

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

    result = product(fracs)

**print**(\*result)

# itertools : Iterables and Iterators

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

x = **int**(**input**())

y = **input**().split()

z = **int**(**input**())

combs = **list**(combinations(y,z))

count = 0

**for** i **in** combs:

**if** 'a' **in** i:

        count += 1

**else**:

**pass**

**print**('%.3f' %(count/**len**(combs)))

# Strings : Alphabet Rangoli

**import** string

**def** print\_rangoli(r):

    s = string.ascii\_lowercase

    pat = [s[r-i-1:r][::-1] + s[r-i-1:r][1:] **for** i **in** **range**(r)]

    pat = pat + pat[:-1][::-1]

**for** i **in** **range**(**len**(pat)):

**print**(\*pat[i].center((r\*2)-1,'-'),sep='-')

*# your code goes here*

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

    n = **int**(**input**())

    print\_rangoli(n)

# 24-11-2020

# python functionals : Validating Email Addresses With a Filter

**import** re

**def** fun(s):

**if** re.match(r'[a-zA-Z0-9-\_]+@[a-zA-Z0-9]+\.[a-zA-Z]{1,3}$',s):

**return** **True**

**else**:

**return** **False**

*# return True if s is a valid email, else return False*

**def** filter\_mail(emails):

**return** **list**(**filter**(fun, emails))

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

    n = **int**(**input**())

    emails = []

**for** \_ **in** **range**(n):

        emails.append(**input**())

filtered\_emails = filter\_mail(emails)

filtered\_emails.sort()

**print**(filtered\_emails)

# Collections : word order

**from** collections **import** Counter, OrderedDict

**class** OrderedCounter(Counter, OrderedDict):

**pass**

d = OrderedCounter(**input**() **for** \_ **in** **range**(**int**(**input**())))

**print**(**len**(d))

**print**(\*d.values())

'''n = int(input())

list1 = []

list2 = []

for i in range(n):

    s = input()

    list1.append(s)

set = set(list1)

list3 = list(set)

for i in list3:

    count = 0

    for j in list1:

        if i == j:

            count += 1

    list2.append(count)

print(len(list2))

for i in list2:

    print(i,end = " ")

    '''