```
except smallest as e:
    print(type(e.args)) #tuple
    for i in e.args:
        print(i, end=" ")
    print("\n")
except largest as e:
    print(type(e.args))
    for in e.args:
        print(i, end=" ")
    print("\n")
```

"Ord" function is used to print ascii value of the word/letter/number

```
print(ord("A"))
```

similarly chr(65) = A

In python, a string of text can be aligned left right & center

.ljust(width)	[left adjust]

```
width = 20
print 'HackerRank'.ljust(width,'-')
```

>>> HackerRank ------

.center(width)

```
>> --- HackerRank ---
```

• rjust(width)

>> ---- Hackerrank.

★ taking input as int in a list

eg.

1 2 3 → single line input

n = list(map(int, input().rstrip().split()))

→ used to convert all the strings to integers

Operators

```
>>> 10/3
3.333333
>>> 10//3 }★
3
```

⇒ Textwrap.

It is a module which provides to convinent functions

(i) textwrap •wrap() wraps a single paragraph in text (a string) so that every line in width character long at most

```
import textwrap
S = "This is a very very very very very long string"
print textwrap.wrap(String, 8)
```

list is returned → ['This is', 'a very', 'very', 'very', 'very', 'long', 'string']

(ii) textwrap.fill() : Wraps a single paragraph in text and returns a single string containing the wrapped paragraph.

```
import textwrap

print textwrap.fill(string, 8)
```

```
This is
a very
very
very
very
long
String
```

- String.count()

```
string.count(substring, [start_index], [end_index])
```

Eg.

```
str = "I live in India, India is a great country"

print (str.count("India"))      # 2
print (str.count("India", 0, 15)) # 1
print (str.count("i"))          # 5
```

Merging 2 Dictionaries:

```
x = {'a':1, 'b':2}
y = {'b':3, 'c':4}
z = {**x, **y}
z = {'c':4, 'a':1, 'b':3}
```

Different ways to test multiple flags at once in Python.

```
x, y, z = 0, 1, 0
```

✓
```
if x==1 or y==1 or z==1:
    print()
```
✓
```
if 1 in (x, y, z):
    print()
```
✓
```
if x or y or z:
    print()
```
✓
```
if any((x, y, z)):
    print
```

Sorting Python dict by value:

```
xs = {'a':4, 'b':3, 'c':2, 'd':1}
```

⇒ `sorted(xs.items(), key=lambda x: x[1])`

`[('d',1), ('c',2), ('b',3), ('a',4)]`

⇒
```
import operator
sorted(xs.items(), key=operator.itemgetter(1))
```

dir(modulename) ⟶ returns all the methods of modules. (func())

Page:
Date:

Conversions :

All these return string which can be formatted

- decimal to binary
  bin(number) ⟷ int('01111', 2)
- hexadecimal ~~(number)~~ => hex(number) ⟷ int('0f', 16)
- octa" = oct(number) ⟷ int('0017', 8)

★ To print upto 6 decimal places float value

needed only

```
st = 1.1234567890
print('{0:.6f}'.format(st))
```

↳ we can also do it like `print("%.2f" %st)`

↳ provides round off value ↓ 1.123

★ Shaping an array list using "numpy" library

Suppose list = [1,2,3,4,5,6,7,8,9]
and we want to print like

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

3x3 array

So we do like

```
import numpy
my_array = numpy.array(list)
my_array.size = (3,3)
print(my_array)
```

Page:
Date:

Sets:

print set ('HackerRank')
set (['a','c','e','H','k','n','r','R'])

★ print set (enumerate (['H','A','c','k','a']))
set ([(4,'a'),(3,'k'),(2,'c'),(0,'H')])

★★ Best Python advantage
eval() → It helps in evaluating any polynomial
Evaluator

★ DefaultDict Tutorial

★ If the keys is not set, it will have default value.

```
from collections import defaultdict
d = defaultdict(list)
d['p'].append("a")
d['s'].append("n")
d['p'].append("l")
for i in d.items()
    print i
```

→ [('p', ['a','l'])
('s', ['n'])

Collections · namedtuple()

* we don't have to use integer indices for accessing members of a tuple.

```
from collections import namedtuple
Point = namedtuple('Point', 'x y')
pt1 = Point(1,2) ⟶ Point(y=2, x=1)
pt2 = Point(3,4) ⟶ Point(x=3, y=4)
product = (pt1.x * pt2.x)+(pt1.y * pt2.y)
print(product) = 11.
```

Collections · OrderedDict

```
from collections import OrderedDict
```

* it remembers the order of the keys that were inserted first. If a new entry overwrites an existing entry, the original position is left unchanged.

```
d = OrderedDict()
d['a'] = 1
d['b'] = 2
d['c'] = 3
d['a'] = 4
print(d)
→ OrderDict([('a',4),('b',2),('c',3)])
```

finding substring :

```
def substring(a):        # a → element
    L=[]
    c=len(a)
    g=0
    for y in range(c):
        for x in range(c):
```

```
        if a[y:(-x)]==' ':
            pass
        else:
            L.append(a[y:(-x)])
    return L
```

Datetime module in python

```
import datetime
from datetime import date
```

- `date(yyyy-mm-dd)` - this function returns a string with passed arguments in order of year.
- `today()` : returns the date of present day in the format yyyy-mm-dd
- ★ `datetime(yyyy, mm, dd)` - object of this date created using which we can perform operations like d1 > d2 or d1-d2, etc.
- ★ `timedelta(days = x)` - used for adding dates like add 25 days to date

Ex.

```
d = date(2011, 4, 7)
p = d + timedelta(days = 25))
```

- `datetime.now(timezone)` - current datetime specification

Ex.

```
current = datetime.datetime.now()
current.year = 2018
current.sec = 4
```

} further methods.

Numpy Module
(Multidimensional Arrays)

```
import numpy as np
a = np.array([2,3,4])  # 1-D
a.dtype  → 'int64'
b = np.array([[2,3],[3,4]], dtype = complex)
        # [[1+0.j, 2+0.j] [3+0.j, 4+0.j]]
c = np.array([(1.5,2,3),(4,5,6)]) # 2-D
        3 rows
np.zeros((3,4))
          ↳ 4 columns
          rows   each columns with 4 elements
np.ones((2,3,4), dtype = np.int16)
          ↓ columns
np.empty((2,3))
np.ones_like(b)
```

★ `np.arange(10,30,5)` # gives [10, 15, 20, 25]

★ `np.linspace(0,2,9)` # 9 numbers from 0 to 2 (↳ included)

`np.sin(x)`

★ `b = np.arange(12).reshape(4,3)` # 2d array

`b.T` ⇝ Transpose

→
```
a = np.array([[4,3,5],[1,2,1]])
b = np.sort(a, axis=1)
```

row wise (by default it is 1 if axis provided) not

Sort by 0 columns

★ numpy[x] → returns whole row
numpy[:,x] → returns whole column

⇒ Product

A * B → Element wise product.

Matrix multiplication → A.dot(B) / np.dot(A,B)

a = np.random.random((2,3))

a.sum() → sum of all elements irrespective of columns & row.

if we provide axis = 0 → sum of each column
= 1 → sum of each row

- clip(——)
- convolve(——) } see this if needed

★ a = np.arange(12).reshape(3,4) # 2D
a.ravel() # converted back to 1D

a.reshape(3,4) — does not modifies a

a.resize((4,3)) — modifies a, change its shape → shape = (4,3)

[[4,5],[6,7][0,1][2,3]]

★ np.vstack((a,b)) → b: [[0,1],[2,3]]
np.hstack((a,b)) a: [[4,5],[6,7]]

↓

[[4 5 0 1] [6 7 2 3]]

np.hsplit(a,3) # split a into 3
np.hsplit(a,(3,4)) # split a after the 3rd and 4th column.

np.vsplit(x,2)}

Page:
Date:

Solving system of equations

$$3x + y + 9z = 39$$
$$x + 2y + 11z = 28$$
$$4x - y + 7z = 32$$

```
a = np.array([[3,1,9],[1,2,11],[4,-1,7]])
b = np.array([39,28,32])
x = np.linalg.solve(a,b)   → gives answer [-,-,-]
```

★ To take a list of N numbers

```
L = list(range(5))   => L = [0,1,2,3,4]
                 ↳ N
```

```
def gcd(x, y):
    while(y):
        x, y = y, x%y
    return x
```

or in python we have direct
`from math import gcd`

Permutation in Python

```
from itertools import permutations
```

★ `print(list(permutations(['1','2','3']))`

⇒ `[('1','2','3'), ('1','3','2'), ('2','1','3'), ('3','2','1')]` all 6 combinations

```
print(list(permutations([['1','2','3'],2]))
[('1','2'),('3','1'), ..... ('3','2')] all 6 combination

print(list(permutations('abc',3))
[('a','b','c'),('a','c','b'), ..... ]
```

=> Cartesian Product

```
from itertools import product

print(list(product([1,2,3],repeat=2)))
[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3), ..... (3,3)]

print(list(product([1,2,3],[3,4])))
[(1,3),(1,4),(2,3),(2,4),(3,3),(3,4)]

A=[[1,2,3],[3,4,5]]
print(list(product(*A))
[(1,3),(1,4),(1,5),(2,3) ..... (3,5)]
```

=> divmod(a,b) => (quotient, remainder)

* If we search for an element in list vs search for an element in set. then it is faster in Set.

Combination in Python:

```
from itertools import combinations

print(list(combinations('12345',2))
```

>>> [('1', '2'), ('1', '3'), ('1', '4'), ('1', '5'), ('2', '3'),
('2', '4'), ('2', '5'), ('3', '4'), ('3', '5'), ('4', '5')]

★ The combination tuples will be produced in sorted order

★ Note :→

l = [(1,2), (2,1), (1,3)]

→ and we want to sort according to 2nd element of every tuple

★ lr = sorted(l, key = lambda x : x[1])

★ → from itertools import combinations_with_replacement
the only difference here is
[('1','1'), ('2','2') ... ('3','3') ...]
are also added in combination

Collections - deque()

★ pop(0) and insert(0,x) in list operations take O(n) but using deque its complexity is O(1)

→ Methods → d = deque() } syntax

- append(x) → add x to the right side of the deque
- appendleft(x)
- clear() → removes all the elements
- count(x)
- extend(iterable) → Extends the right side of the deque by appending elements from the iterable argument

we can also set the maxlen in deque

for eg
```
L = collections.deque(maxlen=2)
L.append('a')
L.append('b')
L.append('c')   → L = ['b', 'c']
```

- extendleft(iterable)
- pop() → right most element
- popleft() → } IndexError is raised if no element is found to be popped.
- remove(value) : removes the first occurance of value
- reverse()
- rotate(n=1) → n steps to the right rotate.
- maxlen → maximum size of a deque

Easy conversion of string to a list

```
s = "dcba"
L = sorted(s)    # ['a', 'b', 'c', 'd']
```

---

any() vs all()

★ list is passed as argument and it should only contain True or False

if all the elements are True it returns True else false → # all(list)
if any element True it returns True else if all elements false → False # any(list)

⇒ Zip() → it returns list of tuples

```
print(zip([1,2,3,4,5,6], [0,9,8,7,6,5,4,3,2,1]))
    → [(1,0), (2,9), (3,8), (4,7), (5,6), (6,5)]
```

★ Truncated to the shortest length.

```
if A = [1,2,3] , B = [6,5,4] , C = [7,8,9]
X = [A] + [B] + [C]
print(zip(*X))   → [(1,6,7), (2,5,8), (3,4,9)]
                    (list)
```

Page:
Date:

★ Basic →

1) if h = [1, 2, 3, 4, 5]
and we do l = h
and pop l then changes will also be there in h
instead do l = list(h) ← creates new list of same elements h

2) for loop has fixed range which is initialized once and cannot be changed
for eg. for x in range(h): h=10
    h = h-1
    print(x)
0
1
2
3
⋮
9

instead to resolve this we use

★ Enumerate

L1 = [`A', `B', `C']
l = list(enumerate(L1)) # [(0, `A'), (1, `B'), (2, `C')]
s1 = "geek"
→ [(0, `g'), (1, `e'), (2, `e'), (3, `k')]
l = list(enumerate(s1, 2))
→ changing start index from 0 to 2 now.

Statistics

⇒ finding Standard deviation
L = list( ) ⟶ [1, 2, 3, ...]
import numpy as np
Sd = np.std(L) ⟶ provides S.d

Redefining "RE"

★ Validating Float number using re

starting symbol ↑ exactly zero or 1 time → means occurs exactly once → end symbol

```
re.match(r"^[+-]?\d*?\.{1}\d{1,}$", k)
```

take only + or - ? | digit | for floating symbol "." | occurance 0 to ∞ times

Symbols → ^ → Start symbol

$ → end symbol

[ ] → indicates set of characters or even the range is been provided like [a-z]

? → occurs exactly zero or once time

\d → for digits

* → occurance zero to infinite times

{ } → defines no. of times, if ',' is provided it provides the range.

+ → like '*' but occurance from one to infinite times (also similar to {1,})

★ \ → any character which is to be checked as it is like in gmail → \@ for at the rate

=> O(sqrt(N)) → to find number is prime or not.

```
def is_prime(n):
    if n == 1:
        return False
    i = 2
    while i*i <= n:
        if n % i == 0:
            return False
        i += 1
    return True
```

⇒ Sieve of Eratosthenes → to find prime in a given range.

Eg 2–25 is the range then

we iterate from 2 to sqrt(25).

for 2 cross all multiples of 2 i.e. 2*2, 3*2, 4*2, ...

for 3 " " " 3 i.e 2*3, 3*3, 4*3 ...

for 4 → already crossed pass for 5 → 2*5, 3*5, 4*5....

Code → Eg. for range 1 to 120.

```
N = 121
is_prime = [1]*N
is_prime[0] = 0
is_prime[1] = 0
def sieve():
    i = 2
    while i*i <= N:
        if is_prime[i] == 0:
            i += 1
            continue
        j = 2*i
        while j < N:
            is_prime[j] = 0
            j += i
        i += 1
sieve()
for i in range(1, N):
    if is_prime[i] == 1:
        print(i)
```

2 3 5 7 .... 109 113.

$O(n \log(\log(n)))$

in a given

2, 4*2, ...
3, 4*3 ...
*5, 3*5, 4*5....

(h))

★ In python we can define a function inside a function.
i.e def ab():
    def cd():
        ——

to call cd() directly without calling ab() directly put global cd just above it.

★ Next Lexicographical permutation algorithm

Eg → 0 1 2 5 5 3 3 0 → 0 1 3 0 2 3 5 5

Steps1→ 0 1 2 [5 5 3 3 0]
↑ pointer
increasing or equal

Step 2→ reverse the red block

0 1 2 [0 3 3 5 5]
↑ pointer

Step3→ now in the red block find the no which is just larger than pointer value i.e 0 1 2 [0 3 3 5 5]
↑ pointer ↑ value searched.

Step4→ Swap both places and thats the result 0 1 3 0 2 3 5 5

O(n).

find this sequence

★ if whole list is in the form of increasing numbers then no further permutation

---

Python sets a recursion limit of about 1000 } Recursive functions

import sys
sys.setrecursionlimit(10000) } can manually raise the limit

Initialize a 4×3 matrix

L = [[0 for x in range(3)] for in range(4)]

★★★ fascinating theory of python.

p = [[]] * 3
p[0].append(1)
p = [[0], [0], [0]]

p = [[] for x in range(3)]
p[0].append(1)
p = [[0], [], []]

Page:
Date:

* proved concept

que = [list([0]*5)[ ] for x in range(2)]

* que[0].append(9) → [[0,0,0,0,0,9], [0,0,0,0,0]]

⇒ Previous lexicographical permutation

1. Initialize i to be len(str)-1
2. Find largest index i such that str[i-1] > str[i] moving i in left direction
3. initialize j to be i-1
4. Find j such that str[j+1] > str[i-1]
5. swap str[j] and str[i-1]
6. Reverse the sub-array starting at str[i]

doubt.

ie str = str[:i] + str[i:][::-1]

* To maintain the order of Set
- Concept use is OrderedDict

```
from collections import OrderedDict
L = [1,2,1,3,1,4]
j = OrderedDict.fromkeys(l).keys()
print(list(j)) → [1,2,3,4]
```

Find every possible combination substring of a string

```
from itertools import combinations
x = 'abc'
L = [''.join(l) for i in range(len(x)) for l in combination(x, i+1)]
→ ['a', 'b', ..... 'abc']
```

)]
0]]

wtation

[i-1] < str[i]

at str[i].
= str[:i] + str[i:]
[::-1]

t]

s()
]

of a string

in combination
(x, i+1)]

1, 4
9

53

compiler problems

from sys import stdin
l = stdin.read().splitlines()
t = int(stdin.readline())

} Rather we have to take whole para as input and iterate through each line.

---

* from itertools import zip_longest
list1 = [1,2,1]
list2 = [2,1,2,3]
[sum(x) for x in zip_longest(list1, list2, fillvalue=0)]
=
[3,3,3,3]

*

[1,2,3] + [4,5,6] = [5,7,9] Motive ↓

i) import numpy as np
list1 = [1,2,3]
list2 = [4,5,6]
list_sum = (np.add(list1, list2)).tolist()

ii) using zip or lambda