```
#Eg. POlymorphism
'''
names = ['alex', 'james', 'Komal', 'Divya']


char = 'j'


def search(name,ch = char): #make deafult argument
    return name[0] == ch


res = list(map(search, names))
print(res)


#it can take an argument which you are passing


'''
# WAP to raise the numbers in list to the power of their indices
'''
nums = [2,3,4,5,6]
#indices = [0,1,2,3,4] #(0 to len(nums)-1)
#indices = range(0, len(nums))
def power(item): #Tuple
    return item[1] ** item[0]
map(power, enumerate(num)) #whenever using enumerate you are passing list of tuple
'''
#or
'''
nums = [2,3,4,5,6]
indices = range(0, len(nums))
def power_(index, item):
    return item ** index

res = list(map(power_,indices , nums))
print(res)

#map(power_, indices, nums) #[1, 3, 16, 125, 1296]
'''

# WAP to convert -ve to +ve numbers in a list
'''
#sum of -ve to +ve numbers
l = [1, 4, 5, -2, 4, -6]
pos = lambda num: num > 0
neg = lambda num: num < 0

pos_num = sum(list(filter(pos, l)))
neg_num = sum(list(filter(neg, l)))

print(pos_num, neg_num)
'''



#Anagram
words = ['eat', 'silent', 'ate', 'tea', 'listen', 'hello']

d = {}

#O/p - {'aet': ['eat', 'ate', 'tea'], 'eilnst': ['silent', 'listen'], 'ehllo': ['hello']}
```

```
#Keys - list are not mutable so typecast it(Key must be immutable)


'''
for word in words:
    key = ''.join(sorted(word))
    if key not in d:
        d[key] = [word]
    else:
        d[key].append(word)
print(d)
'''
#{'aet': ['eat', 'ate', 'tea'], 'eilnst': ['silent', 'listen'], 'ehllo': ['hello']}



# import random
from ast import Lambda
from itertools import count
from operator import index
from re import I, L
from tokenize import group
from turtle import pen
from typing import List

from numpy import number, sort



mylist = [True,False,True]
# # def fun():
# #     return 0.1
# # for i in range(6):
# #     random.shuffle(mylist,fun)
# #     print(mylist[0]^(mylist[1]&mylist[2]))

# print(True^(False and True))
# print(False^(True&True))
# print(True^(True&False))
# print(True&(False^True))
# print(False&(True^True))
# print(True&(False^True))

# # s = ['^','&']

# a,b,c = mylist
# def sym(num1,num2,num3,s):
#     lis = [ ]
#     if s == '^':
#         lis.append(num1^num2)
#         lis.append(num2^num3)
#         lis.append(num1^num3)
#     elif s == '&':
#         lis.append(num1&num2)
#         lis.append(num2&num3)
#         lis.append(num1&num3)
#     elif s == '|':
#         lis.append(num1|num2)
#         lis.append(num2|num3)
#         lis.append(num1|num3)
#     return lis
# and_list = [ i for i in sym(a,b,c,'&') ]
# or_list = [ i for i in sym(a,b,c,'^') ]
# out = [ i^j for i,j,k in zip([c,a,b],and_list,or_list) ]
# out1 = [ i&k for i,j,k in zip([c,a,b],and_list,or_list) ]
# out = []
# for i,j in zip(and_list,or_list):
#     out.append(c&i)
#     out.append(a&i)
#     out.append(b&i)
#     out.append(c^j)
#     out.append(a^j)
#     out.append(b^j)
#     break
# print(out)

# *****************************************************************


# class BankAccount:
#     interest_rate = 0.05
#     def __init__(self,name,balance):
#         self.name = name
#         self.balance = balance
#         self.transactions = [ ]
#         self.transactions.append(f'Initial Balance: {balance}')

#     def deposit(self, amount):
#         self.balance += amount
#         self.transactions.append(f'Deposited amount: {amount}')

#     def withdraw(self, amount):
#         if amount > self.balance:
#             raise Exception('Insufficient Balance')
#         self.balance -= amount
#         self.transactions.append(f'Withdrawl amount: {amount}')

#     def transfer_funds(self, other, amount):
#         if amount > self.balance:
#             raise Exception('Insufficient Funds')
#         other.deposit(amount)
#         self.balance -= amount
#         self.transactions.append('Amount Transfer Done!!')

#     def roi(self):
#         interest_amount = self.balance * self.__class__.interest_rate
#         self.balance += interest_amount
#         self.transactions.append(f'Monthly Interest added: {interest_amount}')

#     def statements(self):
#         for line in self.transactions:
#             print(line)

#         print(f'Total Balance: {self.balance}')



# class SavingsAccount(BankAccount):
#     def __init__(self, name, balance):
```

```
#        super().__init__(name, balance)

#     def withdraw(self, amount):
#        if amount>10000:
#           raise Exception('Exceeded Withdrawl Limit')
#        super().withdraw(amount)

# class SalarayAccount(BankAccount):
#     MAX_DRAFT_AMOUNT = 10000
#     def __init__(self, name):
#        super().__init__(name, balance=0.00)
#        self.taken_draft = 0.00
#        self.is_first_month = True

#     def deposit(self, amount):
#        if self.is_first_month:
#           super().deposit(amount+1000)
#           self.is_first_month = False
#        else:
#           super().deposit(amount)

#     def over_draft(self,amount):
#        if self.taken_draft + amount <= SalarayAccount.MAX_DRAFT_AMOUNT:
#           super().withdraw(amount)
#           self.taken_draft += amount
#        raise Exception('The Total over draft limit exceeded!!')

# class SeniorCitizen(BankAccount):
#     interest_rate = 0.065
#     def __init__(self, name, balance):
#         super().__init__(name, balance)

#     def withdraw(self, amount):
#         if amount > 20000:
#             raise Exception('Your Senior citizen account only supports withdrawl of Rs.20000/-')
#         super().withdraw(amount)

# class GowthamFixedDeposit(BankAccount):
#     interest_rate = 0.1
#     def __init__(self, name, balance):
#         super().__init__(name, balance)

#     def deposit(self,amount):
#         if amount<1000:
#             raise Exception('The minimum deposit is 1000')
#         super().deposit(amount)

#     def withdraw(self, amount):
#         raise Exception('Your Account has no withdraw feature')

# class PenaltyAccount:
#     def __init__(self, penalty_amount):
#         self.penalty_amount = penalty_amount

#     def withdraw(self,amount):
#         super().withdraw(amount)
#         self.balance -= self.penalty_amount
# class PensionAccount(PenaltyAccount,BankAccount):
#     def __init__(self, name, balance, penalty_amount):
#         PenaltyAccount.__init__(self,penalty_amount)
#         BankAccount.__init__(self,name, balance)

# p = PensionAccount('gowtham',10000,500)



# ********************************************************************************
# ele = {'b':2,'a':3,'c':5}
# def vals(d):
#     return d.values()
# sot = sorted(ele, key=vals)
# print(sot)

# ********************************************************************************
# WAP to consecutive longest element
# s = 'abbccccdddddddeeeeeeedeeeeeca'
# lis = [ ]
# for i in set(s):
#     st=''
#     for j in s:
#         if i==j:
#             st+=j
#         else:
#             if st:
#                 lis.append(st)
#                 st=''
#     if st:
```

```
#       lis.append(st)

# print(max(lis,key=len))
# ******************************************************************************
# a='aabbbbaaccccc'
# col=[]
# for i in set(a):
#         res=""
#         for j in a:
#                 if i==j:
#                         res+=j
#                 else:
#                         col.append(res)
#                         res=""
#         col.append(res)
# val=len(max(col))
# [print(k) for k in col if len(k)==val]


# ******************************************************************************
names = ['apple','google','yahoo','amazon','facebook','instagram','microsoft','zomato']
# Sort the above list based on first character in each element
first_char = sorted(names)

# ******************************************************************************
# Sort the list based on last character in each element
last_char = sorted(names,key=lambda i:i[-1])

# ******************************************************************************
```

```
prices = {'ACME':45.23,'AAPL':612.78,'IBM':205.55,'HPQ':37.20,'FB':10.75}

# Sort the dictionary based on the value

val_dict = dict(sorted(prices.items(),key=lambda i:i[-1]))


# ***************************************************************************

# Sort the dictionary based on key

key_dict = dict(sorted(prices.items(), key=lambda i:i[0]))


# ***************************************************************************

# Sort the dictionary based on the length of key

key_len_dict = dict(sorted(prices.items(), key= lambda i: len(i[0])))


# ***************************************************************************

# Sort the dictionary based on the length of value

val_len_dict = dict(sorted(prices.items(), key= lambda i: len(str(i[-1]))))


# ***************************************************************************

# Sort the dictionary based on the last char of key/value

key_last_char_dict = dict(sorted(prices.items(), key= lambda i: i[0][-1]))

val_last_char_dict = dict(sorted(prices.items(), key= lambda i: str(i[-1])[-1]))


# ***************************************************************************

# Sort the dictionary based on first char of key/value

key_first_char_dict = dict(sorted(prices.items(), key= lambda i: i[0][0]))

val_first_char_dict = dict(sorted(prices.items(), key= lambda i: str(i[-1])[0]))


# ***************************************************************************

# WAF that accepts two strings and returns True if the two strings are anagrams of each other

def is_anagram(str1,str2):
    out = []
    for i in str1:
        if len(str1)==len(str2) and i in str2:
            out.append(1)
        else:
            out.append(0)
    return all(out)

# ****************************************************************************
# Grouping anagrams
words = ['eat','ate','tea','hello','silent','listen']
w=words
group_dict = { }
for i in words:
    d = [ ]
    for j in w:
        if is_anagram(i,j)==True:
            d.append(j)
    group_dict[i]=group_dict.get(i,[])+[d]
# print(group_dict)

# ****************************************************************************
# COMPREHENSIONS
# ********************
# Build a list of prime numbers
prime_list = [ i for i in range(1,51) if all(i%j!=0 for j in range(2,i)) ][1:]
# print(prime_list)

# ****************************************************************************
# Reverse the item of the list if the item has odd length
name_s = ['apple', 'google', 'yahoo', 'amazon', 'yelp', 'flipkart', 'gmail', 'facebook', 'instagram',
'microsoft', 'zomato']
rev_names = [ i[::-1] if len(i)%2!=0 else i for i in name_s ]
# print(rev_names)

# *******************************************************************************
# Create a lambda function
num = lambda i:i+15
# print(num(10))

# *******************************************************************************
# Lambda function to square and cube any number
square = lambda i: i**2
cube = lambda i: i**3
# print(square(2))
# print(cube(3))



# *******************************************************************************
# FILTER
# *******


# WAP to filter only even numbers from 1 to 50
def eve(num):
   return (num if num%2==0 else None)

eve_filter = list(filter(eve,range(1,51)))
# print(eve_filter)
# ******************************************************************************
# WAP to filter only even numbers from 1 to 50
def odd(num):
    return (num if num%2!=0 else None)

odd_filter = list(filter(odd,range(1,51)))
# print(odd_filter)

# ******************************************************************************
def eve_len(lis):
    if len(lis)%2==0:
        return lis
    else:
        None
# print(list(filter(eve_len, name_s)))

# ******************************************************************************
emp_name = ['laura','steve','bill','james','bob','greig','scott','alex','ive']
def vow(st):
    if st[0] in 'aeiouAEIOU':
        return st
# print(list(filter(vow, emp_name)))

# ******************************************************************************
numbers = [-2,-1,0,1,2]
def pos_num(numb):
    return (numb if numb>0 else None)
# print(list(filter(lambda x:x if x>0 else None,numbers)))
# ********************************************************************************

# Filter prime numbers in range 1 to 50

print(list(filter(lambda x:x if all(True if x%i!=0 else False for i in range(2,x)) else None,list(range(1,51)))))



# print(list(map(lambda x:x*2,filter(lambda x:x if all(True if x%i!=0 else False for i in range(2,x)) else None,list(range(1,51))))))


# ********************************************************************************

# MAPS

# ******


# WAP to square and cube every integer in the list

lis = [1,2,3,4,5,6]

sqare = map(lambda i: i**2, lis)

cobe = map(lambda i:i**3, lis)



# WAP to find if a string starts with given character or not

string = 'Hello World!'

ch = 'H'

find_char = map(lambda i: True if i.startswith(ch) else False, [string])



# WAP to convert -ve to +ve numbers ina list

l = [-1,-2,-3,-4,-5]

pos_n = map(lambda i: abs(i), l)
```

```
# WAP to raise the numbers in list to the power of their indices

l1 = [1,2,3,4,5]

pow_num = map(lambda i:i[1]**i[0], enumerate(l1))


# ***************************************************************************

# GENERATOR - ASSIGNMENT

# **********************

#


a = [9,5,2,7,-1,6,7,8,-1]


def pivot_sort(lst,ele):

    sti = ''

    for i in lst:

        sti+=str(i)+','

    nums = sti.split(str(ele))

    out_list = [ ]

    for i in nums:

        li = [ ]

        if i:

            for j in i.split(','):

                if j:

                    li.append(int(j))

        out_list.extend(sorted(li))

    for j in range(len(lst)):

        if lst[j]==ele:

            out_list.insert(j,ele)
    return out_list


print(pivot_sort(a,-1))
```