```python
def fact():
    def factorial(n):
        if n==1: return 1
        else: return n*factorial(n-1)
    n=int(input('Enter the factorial number:'))
    print(factorial(n))

def armstrong():
    num = input('Enter the number:')
    l = len(num)
    sum = 0
    for i in range(l):
        n = int(num[i]) ** l
        sum = sum + (int(num[i]) ** l)
    print(sum)
    if sum == int(num):
        print('The given number is Armstrong.')
    else:
        print('The given number is not Armstrong.')

def pattern2():
    n = int(input())
    for i in range(1, n + 1):
        for l in range(n, i, -1):
            print(' ', end=' ')
        for j in range(1, i + 1):
            print('*', end=' ')
        for j in range(2, i + 1):
            print('*', end=' ')
        print()
    for i in range(1, n + 1):
        for m in range(1, i):
            print(' ', end=' ')
        for k in range(i, n, 1):
            print('*', end=' ')
        for k in range(i - 1, n, 1):
            print('*', end=' ')
        print()
    return

def pattern1():
    n = int(input())
    for i in range(1, n + 1):
        for l in range(n, i, -1):
            print(' ', end=' ')
        for k in range(1, i + 1):
            print(k, end=' ')
        for j in range(2, i + 1):
            # print(j,end='')
            print(j + i - 1, end=' ')
        print()
    return

def larger_of_3_integer():
    lis = []
    for i in range(3):
        lis.append(input())
    lis.sort()
    print(lis[-2])
    return(lis[-2])
```

```
def atm():
    details = [['pranathi', 1234, 1000], ['malli', 3456, 2000],
['parimala', 2004, 3000]]

    def default():
        print('Invalid input')

    def balance(current):
        print(details[current][2])

    def withdraw(current):
        amount = int(input('Enter the withdraw amount'))
        if amount > details[current][2]:
            print('Insufficient balace')
        else:
            total = details[current][2] - amount
            details[current][2] = total
            print('Your balance amount is ', total)

    def deposit(current):
        amount = int(input('Enter the deposit amount'))
        total = details[current][2] + amount
        details[current][2] = total
        print('Your balance is ', total)

    def reset(current):
        new_pass = int(input('Enter the new password'))
        re_pass = int(input('Re-enter the password'))
        if new_pass == re_pass:
            details[current][1] = new_pass
            print('Password updated')
        else:
            print('Type correct password')
        return details

    def main():
        name = input('Enter the user name')
        for i in range(len(details)):
            if name == details[i][0]:
                current = i
                for j in range(3, 0, -1):
                    password = int(input('Enter the password'))
                    if password == details[current][1]:
                        def switch_case(self):
                            print(
                                '\nEnter 1 for checking balance \nEnter 2
for Withdrawing amount \nEnter 3 for Depositing amount \nEnter 4 to reset
password \nEnter 0 to complete the process')
                            n = int(input('\nEnter a function name'))
                            switch_data = {1: balance, 2: withdraw, 3:
deposit, 4: reset}
                            result = switch_data.get(n, default)
                            if n == 0:
                                print('Thank you !!! Process completed')
                                return main()
                            elif n == 4:
                                details = result(current)
                                main()
                            else:
                                result(current)
                                return switch_case(current)
                        switch_case(current)
                    else:
                        print('Password incorrect, try again !')
                        print('You have ', j - 1, 'chances left.')
                        if j - 1 == 0:
                            print('Account is blocked')
                            return main()
            elif name == 'exit':
                    exit()
        else:
            print('Invalid input')

    main()

def string_pattern():
    name = input('Enter the string:')
    for i in range(len(name)):
        for k in range(0, i + 1):
            print(name[k], end=' ')
        print()
    return


n=5

for i in range (1,n+1):

    if i==1 or i==n:print("* "*n)

    else:print("","  "(n-3)," *")
#

name=input('Enter the string')
while True:
    n = int(input('Enter 0 for left rotation and 1 for right rotation'))
    if n == 0:
        num = int(input('Enter number of times of rotation'))
        for i in range(num):
            name=name[1:]+name[0]
        print(name)
        break
    elif n == 1:
        num = int(input('Enter number of times of rotation'))
        for j in range(num):
            name=name[-1]+name[:-1]
        print(name)
        break
    else:
        print('Invalid input, please enter 0 or 1')
#

class node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None
    def insert_at_end(self,data):
        newNode=node(data)
```

```python
        if self.head is None:
            self.head=newNode
        else:
            current=self.head
            while current.next is not None:
                current=current.next
            current.next=newNode
    def insert_at_beg(self,data):
        newNode=node(data)
        if self.head is None:
            self.head=newNode
        else:
            newNode.next=self.head
            self.head=newNode
    def insert_at_pos(self,data,pos):
        newNode=node(data)
        if self.head is None:
            self.head=newNode
        elif pos ==1:
            newNode.next = self.head
            self.head = newNode
        else:
            current=self.head
            count=0
            while current:
                count += 1
                if count == pos-1:
                    newNode.next=current.next
                    current.next=newNode
                    break
                current = current.next
            if count<pos-1:
                self.insert_at_end(data)
    def insert_after_data(self,key,data):
        newNode=node(data)
        if self.head is None:
            self.head=newNode
        else:
            current=self.head
            count=0
            while current:
                if current.data == key:
                    count += 1
                    newNode.next=current.next
                    current.next=newNode
                current=current.next
            if count==0:
                print('Data not found')
    def insert_before_data(self,key,data):
        newNode=node(data)
        current=self.head
        if self.head is None:
            self.head=newNode
        elif current.data == key:
            self.insert_at_beg(data)
        else:
            count=0
            while current.next:
                if current.next.data == key:
                    count+=1
                    newNode.next = current.next
                    current.next = newNode
                    break
                current=current.next
            if count==0:
                obj.insert_at_end(data)

    def delete_at_beg(self):
        if self.head is None:
            print('Lnked list is empty')
        else:
            current = self.head
            self.head = self.head.next
            current.next = None
    def delete_at_end(self):
        current=self.head
        if self.head is None:
            print('Lnked list is empty')
        elif current.next is None:
            current=None
        elif current.next.next is None:
            current.next=None
        else:
            current=self.head
            while current.next.next:
                current=current.next
            current.next=None
    def delete_by_value(self,data):
        current=self.head
        if self.head is None:
            print('Data not found')
        elif current.data == data:
            self.delete_at_beg()
        else:
            while current.next:
                if current.next.data == data:
                    current.next=current.next.next
                    current = current.next
                    break
                current=current.next

    def display(self):
        current=self.head
        while current:
            print(current.data,end=' ->')
            current=current.next
obj=linkedlist()
obj.insert_at_beg(5)
obj.insert_at_end(4)
obj.insert_before_data(4,0)
obj.insert_at_beg(6)
obj.insert_at_beg(2)
obj.insert_after_data(6,7)
obj.insert_at_pos(9,10)
#obj.delete_at_end()
#obj.delete_at_beg()
obj.delete_by_value(9)
obj.display()
```

P2

```
for i in range(10,0,-1):
    print(i, end=" ")


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

for i in range(2):
    print('P',end=' ')
    for j in range(6):
        print('S')
        if j==1:
            break


add=lambda a,b:print(a+b)
print(add(2,3))

numbers=[1,2,3,4]
squared=list(map(lambda x:x**2,numbers))
print(squared)

def fun(n):
    if n==4:
        return 5
    print(fun(n+1))
fun(1)
```

#P3

```
def num(n):
    print(n,end=' ')
    if n == 10:
        return
    num(n+1)

num(1)



#generator function
def count(n):
    i=1
    while i<=n:
        yield i
        i+=1
print(list(count(5)))




#decorator function
def my_decorator(func):
    def wrapper():
        print('Something is happening before the function')
        func()
        print('Something is happening after the function')
    return wrapper
@my_decorator
def say_hello():
    print('Hello')
#my_decorator(say_hello())
say_hello()


#closure function
def outer(x):
    def inner(y):
        return x+y
    return inner
closure=outer(10)
result=closure(5)
print(result)

#Arguments
def fun(**a):
    print(a)
fun(name='lucky',age=19)

#Arbitary keyword arguments
def fun(*a):
    print(a)
fun(1,2,3)
```

#P4

```
#Exception handling
a=10
b=0
try:
    print(a/b)
except ZeroDivisionError:
    print('AAA')
except Exception as e:
    print(e)

else:
    print('hi')
finally:
    print('finally')

from itertools import combinations
l=[1,2,3,4,5]
print(list(combinations(l,3)))


from itertools import permutations
l=[1,2,3,4,5]
print(list(permutations(l,2)))
```

#P5

```
#nested class
class A:
    class B:
        class C:
            a=10
obj=A().B().C()
"""obj=A()
obj1=obj.B()
obj2=obj1.C()
print(obj2.a)"""
print(obj.a)


class A:
    def __init__(self):
        print('a')
        self.b=self.B()
    class B:
        def __init__(self):
            print('b')
            self.c=self.C()
        class C:
            def __init__(self):
                print('c')
A()


class Student:
    def __init__(self, *marks):
        self.m1 = marks[0]
        self.m2 = marks[1]
        self.m3 = marks[2]
        self.m4 = marks[3]

    def calculate_total(self):
        self.total_marks = self.m1 + self.m2 + self.m3 + self.m4

    def rank(self):
        l = []
        for i in self:
            l.append(i.total_marks)
        print(l)


# a = [100, 100, 100, 100]
# b = [90, 90, 90, 90]
# c = [80, 80, 80, 80]

malli = Student(100,100,100,100,100)
ramya = Student(90,90,90,90,90)
vaishu = Student(80,80,80,80,80)

malli.calculate_total()
ramya.calculate_total()
vaishu.calculate_total()
#Student.rank(vaishu)
Student.rank(malli, ramya, vaishu)
```

#P6

```
class A:
    def duster(self):
        print('duster')
```

```
class B(A):
    def projector(self):
        print('projector')
apple=A()
ball=B()
ball.projector()
```

#P7

```
#frozenset
a={1,2,3}
a.add(5)
print(a)
a=frozenset(a)
#a.add(7)
print(a)
print(type(a))

#method overwriting (same method but different with respect to object)
class A:
    def ab(self):
        print('trainer')
class B:
    def ab(self):
        print('friend')
obj=B().ab()
obj1=A().ab()

#method overloading depends on number of arguments passed to function
class A:
    def ab(self,a=0):
        print('classes')
    def ab(self,a=0):
        print('exam')
obj=A()
obj.ab()


class a:
    def ab(self,python,java=0):
        if java==0:
            print('trainer',python)
        elif python and java:
            print('trainer',python,java)
obj=a().ab('python','java')

#duck typing
class cat:
    def sound(self):
        print('meow')
class dog:
    def sound(self):
        print('bow')
class duck:
    def sound(self):
        print('buck')
def make_sound(self):
    self.sound()
# c=cat()
# d=dog()
```

```python
# du=duck()
# make_sound(d)
# make_sound(c)
for obj in cat(),dog(),duck():
    obj.sound()

# '__' indicates private
# '_' indicates protected, in python it is also treated as public

# class A:
#     ___a=10
# print(A().___a)

#data encapsulation
class A:
    def krishna(self):
        self.__display()
    def __display(self):
        print('display')
A().krishna()
```

```python
#Data abstraction
from abc import ABC, abstractmethod
class Base(ABC):
    @abstractmethod
    def display(self):
        pass

from P8 import *
class A(Base):
    def display(self):
       print('hi')

from P9 import *
A().display()
```

#P8

```python
#linear search
l=list(map(int,input("Enter the list value").split(" ")))
key=int(input("enter the key value:"))
for i in range(len(l)):
    if l[i]==key:
        print(f"value found at {i}th position in the list")
        break
    # elif l[i]!=key:
    #     print("not found")
if key not in l:
    print("value not found")

#binary search
det=list(map(int,input("Enter the list value").split(" ")))
key=int(input("enter the key value:"))
l=0
r=len(det)-1
while l<=r:
    mid=(l+r)//2
    if det[mid]==key:
        print(mid)
        break
    elif det[mid]<key:
        l=mid+1
    else:
        r=mid-1
if key not in det:
    print('Value not found')

#bubble sort
l=[1,5,8,2,3]
n=len(l)
for i in range(n):
    for j in range(0,n-i-1):
        if l[j]>l[j+1]:
            l[j],l[j+1]=l[j+1],l[j]
print(l)

#
l=[1,5,6,8,2,0]
n=len(l)
for i in range(n):
    for j in range(n):
        if l[i]<l[j]:
            l[i],l[j]=l[j],l[i]
print(l)

#selection sort
l=[1,5,8,3,9]
n=len(l)
for i in range(n):
    min=i
    for j in range(i+1,n):
        if l[min]>l[j]:
            min=j
    l[i],l[min]=l[min],l[i]
print(l)
```

#P9

```
#insertion sort
l=[2,5,0,7,3]
n=len(l)
for i in range(1,n):
    key=l[i]
    j=i-1
    while j>=0 and l[j]>key:
        l[j+1]=l[j]
        j=j-1
    l[j+1]=key
print(l)

#merge sort
def merge(lis):
    if len(lis)>1:
        mid=len(lis)//2
        left=lis[:mid]
        right=lis[mid:]
        merge(left)
        merge(right)
        l=r=k = 0
        while l < len(left) and r < len(right):
            if left[l] > right[r]:
                lis[k] = right[r]
                r += 1
            else:
                lis[k] = left[l]
                l += 1
            k = k + 1
        while l < len(left):
            lis[k] = left[l]
            l += 1
            k += 1
        while r < len(right):
            lis[k] = right[r]
            r += 1
            k += 1
lis=[5,9,0,3,4]
merge(lis)
print(lis)

#linked list
class node:
    def __init__(self,data):
        self.data=data
        self.next=None
class linkedlist:
    def __init__(self):
        self.head=None
    def insert_at_end(self,data):
        newNode=node(data)
        if self.head is None:
            self.head=newNode
        else:
            current=self.head
            while current.next is not None:
                current=current.next
            current.next=newNode
    def insert_at_beg(self,data):
        newNode=node(data)
        if self.head is None:
            self.head=newNode
        else:
            newNode.next=self.head
            self.head=newNode
    def display(self):
        current=self.head
        while current:
            print(current.data,end=' ->')
            current=current.next
obj=linkedlist()
obj.insert_at_beg(5)
obj.insert_at_end(4)
obj.insert_at_beg(6)
obj.insert_at_beg(2)
obj.display()


#
```

```python
i=2
def fun(name,i):
    password = input('Enter the password:')
    if password == user[name]:
        print('Access accepted')
        return
    else:
        #i=i-1
        print('Password incorrect, try again !')
        print('You have ', i, 'chances left.')
        if i== 0:
            print('Account is blocked')
            return
        return fun(name,i-1)
user={'pranathi':'1234','ramya':'1023','poornima':'5678'}
a=input('Enter the user name:')
if a in user.keys():
    fun(a,i)
else:
    print('Username not found.')




#

class A:
    def __init__(self):
        print('A')
        super().__init__()
class B:
    def __init__(self):
        print('B')
class C(A,B):
    def __init__(self):
        print('C')
        super().__init__()
        #super().__init__()
C()
```