1. Bubble sort

Q. Assending order of list without using inbuit function

```
a= [10, 30, 50, 90, 20]
for i in range (len(a)):
    for j in range (i+1, len(a)):
        if a[i] > a[j]:
            a[i], a[j] = a[j], a[i]

print (a)
```

3.Amstrong number

Q.2 Python Program to check Amstrong number.

```
num = int (input ("Enter the number :"))

sum = 0

temp = num
while temp >0 :
    digit = temp %10
    sum += digit **3
    temp //= 10

if num == sum :
    Print (num, "is an armstrong number")
else :
    Print (num, "is not an amstrong number")
```

O/P - 1

Enter the number : 663
663 is not an armstrong number

O/P - 2

Enter the number : 407
407 is an armstrong number

4. Prime number

Q. Python Program to print all prime number is an intervals.

```
lower = 900
upper = 950
Print ("Prime number between", lower "and" uper "are")
for num in range (lower, upper+1):
    if num > 1 :
        for i in range (2, num):
            if (num % i) == 0 :
                break
        else :
            Print (num)
```

O/P -

Prime number between 900 & 950 are
907
911
919
929
937
941
947

5. Factorial Number

Q.1 Python program for factorial of a number with and without recursion.

→ with recursion :-

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

n= int(input("Enter the number : "))
result = fact(n)
Print ("Factorial of ",n,"is",result)
```

O/P - Enter the number : 5
Factorial of 5 is 120

Without recursion -

```
n= int(input("Enter the number : "))
fact = 1
if n<0 :
    print ("factorial doesn't exist for negative number")
else :
    for i in range (1, n+1):
        fact = fact * i
    Print("factorial of", n, 'is' fact)
```

O/P - Enter the number 5
factorial of 5 is 120

2. Print the duplicate from a list

Q. Python program to print duplicate from a list integers.

→

```
l= [1,2,2,3,4,5,5,6,7]

l2= [ ]
for i in l :
    n= l.count[i]
    if n>1 :
        l2.append(i)
print (l2)
```

O/P - [2,2,5,5]

To remove duplicate from the list

```
l= [1,2,2,3,4,5,5,6,7]

a= list(set(l))
print (a)
```

O/P-
[1,2,3,4,5,6,7]

6. Fibonacci number

② Function for fibonacci number.

```
def Fibonacci (n) :

    if n<0 :
        print ("incorrect input")

    elif n==0 :
        return 0

    elif ~~n==1~~ n==1 or n==2
        return 1

    else :
        return Fibonacci (n-1) + Fibonacci (n-2)

a = Fibonacci (9)
print (a)
```

O/P -
34

→ ② Fibonacci number

```
def Fibonacci (n) :
    a=0
    b=1

    if n<0 :
        print ("incorrect input")

    elif n==0 :
        return 0

    elif n==1 :
        return b

    else :
        for i in range (1, n):
            c= a+b
            a=b
            b=c
        return b

Print (fibonacci (9))
```

O/P -
34

7. Largest and second largest number

Q Second largest number in list

→ ① l = [1,2,3,4,5,6]
l.sort()
print("second largest no.", l[-2])

O/P - 5

② l = [10,20,30,40,50,60]
largest = 0
for i in l:
if i > largest :
largest = i
second largest = 0
~~for i in l~~ :
for i in l :
if i ≠ largest and i > second largest:
second largest = i
print(second largest)

O/P -
50

8. String occurrence

input - akshay
O/P a = 2
k = 1
s = 1
h = 1
y = 1

Program

s = input("Enter name")
d = { }
for x in s:
if x in d.keys()
d[x] = d[x] + 1
else :
d[x] = 1
for k, v in d.items():
print("{} = {} times".format(k,v))

```
ip = "H#E@L$@L O#WO$R%LD"
op=[]
temp=[]
for i in ip:
    if i.isalpha():
        op.append(i)
    else:
        temp.append(i)
print(op)
print(temp)

# output
# ['H', 'E', 'L', 'L', 'O', 'W', 'O', 'R', 'L', 'D']
# ['#', '@', '$', '@', ' ', '#', '$', '%']
```

```
 r='Hello Google'
# r1=r.split()            ##reversing method
# op=[]
# for i in r1:
#     op.append(i[::-1])
# print(' '.join(op))


# output-olleH elgooG
```

```
# a='aaaaabbbbbcccccdddddd'
# l={}
# for i in a:
#     if i in l:
#         l[i]=l[i]+1
#     else:
#         l[i]=1
# output=''
# for k,v in l.items():
#     output=output+k+str(v)
# print(output)

# output=a5b5c5d6
```

```
a='aaaaabbbbbcccccdddddd'
op=''
for i in a:
    if i not in op:
        op=op+i
print("dublicate remove:",op)

output= dublicate remove: abcd
```

```
#How to check vowels in giving string in the from of count:

# a=input('enter string:')
# b=set(a)
# vowels='a','e','i','o','u'
# cnt=0
# for i in b:
#     if i in vowels:
#         cnt=cnt+1
# print('vowels:',cnt)
```

calculating the wight space by using this

```
 a1='rohit       bundile'                    # calculating the wight space by using this
method
# cnt=0
# for i in a1:
#     if i.isspace():
#         cnt=cnt+1
# print('Total cnt:',cnt)
```

```
Dictionary problem:
# l=['a','b','c','d','e']
# l2=[1,2,5,4]
# l3={l[i]:l2[i] for i in range(len(l2)) }        #take a two variable convert into
dictionary
# print(l3)
```

```
#2nd Aporoch:
# l4=dict(zip(l,l2))                          #take a two variable convert into dictionary
# print(l4)
```

```
decorator : Decorator is a function which can take function as argument add sum function
# and return the modified function with extended functionality
# def decore1(func):
#     def inner(x):
#         y=10
#         add=x+y
#         return add
#     return inner
#
# @decore1
# def decor(x):
#     # print("x value ",x)
#     return x
# xy=decor(5)
# print(xy)
```

```
#Generator : Generator is function which Responsible to generate a sequence of values we
can write_
#_generator function just like ordinary function  but its used to yield keyword for
returning the value.

# def gen(n):
#     i=0
#     while i<=n:
#         yield i
#         i=i+1
# a=gen(6)
# print(a)
# for i in a:
#     print(i)
```

Shuffle Program

input=[-5,-7,-2,2,5,-6,0,6,0,0]

Output=[--5,-7,-2,-6,2,5,6,0,0,0]

```
shuffle program

import random

def shuffle(l):
        random.shuffle(l)
        return l
l=[1,2,3,4,5]
print(shuffle(l))
```

```
lst=[-5,-7,-2,0,5,8,0,9,3,-8]
positive_number=sorted([num for num in lst if num>0])
negative_number=sorted([num for num in lst if num<0])
zeros=[num for num in lst if num==0]
print(positive_number+negative_number+zeros)

output=[3, 5, 8, 9, -8, -7, -5, -2, 0, 0]
-----------------------------------------------------------
```

Class Method and Static method and Instance method

```
Class method static method and Instance method


class Myclass:
    value=10

    def __init__(self,value1):
        self.value1=value1

    def instance_method(self):
        print("Instance method",self.value1)

    @classmethod
    def class_method(cls):
        print("class method",cls.value)

    @staticmethod
    def static_method():
        print("static method")

obj=Myclass(50)

obj.instance_method()
Myclass.class_method()
obj.static_method()
--------------------------------------------------
```

Bracket question

```
Two brackets are considered to be a matched pair if the an opening
bracket (i.e., (, [, or {) occurs to the left of a closing bracket
(i.e., ), ], or }) of the exact same type. There are three types
of matched pairs of brackets: [], {}, and ().
A matching pair of brackets is not balanced if the set of brackets
it encloses are not matched
Ex
Balanced {{[()]}}
Not Balanced {{[()]}}}}


def balanced_brackets(s):
        a=[]

        brackets={')':'(','}':'{',']':'['}

        for char in s:
                if char in '([{':
                        a.append(char)

                elif char in '})]':
                        if not a:
                                return 'NO'

                        top=a.pop()
                        if brackets[char]!=top:
                                return "NO"
        return "YES" if not a else "NO"

print(balanced_brackets("{{[()]}}"))
print(balanced_brackets("{{[()]}}}}"))
```

```python
def is_balanced(string):
    stack = []
    opening_brackets = "([{"
    closing_brackets = ")]}"
    bracket_pairs = {')': '(', ']': '[', '}': '{'}

    for char in string:
        if char in opening_brackets:
            stack.append(char)
        elif char in closing_brackets:
            if not stack or stack[-1] != bracket_pairs[char]:
                return "NO"
            stack.pop()

    return "YES" if not stack else "NO"

balanced_string = "{{[()]}}"
not_balanced_string = "{{[()]}}}}"
print(is_balanced(balanced_string))  # Output: YES
print(is_balanced(not_balanced_string))  # Output: NO
```

Inheritance concept

```
Multiple inheritance

class Parent1:
    def __init__(self,name):
        self.name=name
        print("parent1 class constructor")

    def method1(self):
        print("parent class 1=",self.name)


class Parent2:
    def __init__(self,age):
        self.age=age
        print("parent2 class constructor")

    def method2(self):
        print("parent class 2=",self.age)

class Child(Parent1,Parent2):
    def __init__(self,name,age,place):
        Parent1.__init__(self,name)
        Parent2.__init__(self,age)
        self.place=place
        print("child class constructor")

    def method3(self):
        print("child class=",self.place)

obj=Child("akshay",28,"solapur")
obj.method1()
obj.method2()
obj.method3()
```

```
def square_box1(sentense):
    words=sentense.split()
    max_len=max(len(word) for word in words)
    print("" (max_len +3))
    for word in words:
        p=max_len-len(word)
        print("*" + " "+ word + " " * p + "*")
    print("*" * (max_len + 3))
sentense="This is for python test"
square_box1(sentense)
```

10:41

```
*********
* This  *
* is    *
* for   *
* python*
* test  *
*********
```

10:41

3. Inverted Pyramid Pattern:

python Copy code

```python
def inverted_pyramid(rows):
    for i in range(rows, 0, -1):
        print(" " * (rows - i) + "*" * (2 * i - 1))

inverted_pyramid(5)
```

Output:

markdown Copy code

```markdown
*********
 *******
  *****
   ***
    *
```

Regene

1. Pyramid Pattern:

python Copy code

```python
def pyramid_pattern(rows):
    for i in range(1, rows + 1):
        print(" " * (rows - i) + "*" * (2 * i - 1))

pyramid_pattern(5)
```

Output:

markdown Copy code

```markdown
    *
   ***
  *****
 *******
*********
```

Regen

2. Right Triangle Pattern:

python Copy code

```python
def right_triangle(rows):
    for i in range(1, rows + 1):
        print("*" * i)

right_triangle(5)
```

Output:

markdown Copy code

```markdown
*
**
***
****
*****
```

Regen

5. Hollow Square Pattern:

python Copy code

```python
def hollow_square(rows):
    for i in range(1, rows + 1):
        if i == 1 or i == rows:
            print("*" * rows)
        else:
            print("*" + " " * (rows - 2) + "*")

hollow_square(5)
```

Output:

markdown Copy code

```markdown
*****
*   *
*   *
*   *
*****
```

4. Diamond Pattern:

python Copy code

```python
def diamond_pattern(rows):
    for i in range(1, rows + 1):
        print(" " * (rows - i) + "*" * (2 * i - 1))
    for i in range(rows - 1, 0, -1):
        print(" " * (rows - i) + "*" * (2 * i - 1))

diamond_pattern(5)
```

Output:

markdown Copy code

```markdown
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

Regener