Hashing

1] Implementation of Chaining:

```
class MyHash:
    def __init__(self,b):
        self.BUCKET=b
        self.table=[[] for x in range(b)]

    def insert(self,x):
        i=x%self.BUCKET
        self.table[i].append(x)

    def remove(self,x):
        i=x%self.BUCKET
        if x in self.table[i]:
            self.table[i].remove(x)

    def search(self,x):
        i=x%self.BUCKET
        return x in self.table[i]

h=MyHash(8)
h.insert(79)
h.insert(71)
h.insert(9)
h.insert(30)
h.insert(350)
print(h.search(9))
h.remove(9)
print(h.search(9))
```

OUTPUT:

True

False

2] Implementation Of Open Addressing:

```
class MyHash:
    def __init__(self,c):
        self.cap=c
        self.table=[-1]*c
        self.size=0

    def hash(self,x):
        return x%self.cap

    def search(self,x):
        h=self.hash(x)
        t=self.table
        i=h
        while t[i]!=-1:
            if t[i]==x:
                return True
            i=(i+1)%self.cap
            if i==h:
                return False
        return False

    def insert(self,x):
        if self.size==self.cap:
            return False

        if self.search(x)==True:
            return False

        i=self.hash(x)
        t=self.table
        while t[i] not in (-1,-2):
            i=(i+1)%self.cap

        t[i]=x
        self.size+=1
        return True

    def remove(self,x):
        h=self.hash(x)
        t=self.table
        i=h
        while t[i]!=-1:
            if t[i]==x:
                t[i]=-2
```

```
                return True
            i=(i+1)%self.cap
            if i==h:
                return False
        return False


h=MyHash(8)
h.insert(79)
h.insert(71)
h.insert(9)
h.insert(30)
h.insert(350)
print(h.search(9))
h.remove(9)
print(h.search(9))
```

OUTPUT :

True

False

3] Set In Python :

```
"""A set is an unordered collection of items. Every set element is unique
(no duplicates) and must be immutable (cannot be changed). However,
a set itself is mutable. We can add or remove items from it."""
#creation
print("************************ Creation ******************")

s1={10,20,30}
print(s1)

s2=set([20,30,40])
print(s2)

s3={}
print("expected type set ", type(s3))

s4=set()
print(type(s4))
print(s4)
print()


#insertion
print("********************** Insertion **********************")

s={10,20}
s.add(30)
print(s)

s.add(30)  # add duplicate item
print(s)

s.update([40,50])
print(s)

s.update([60,70],[80,90])  #insert multple list
print(s)
print()


#remove
print("********************** Remove **********************")
"""
The discard() method removes the specified item from the set. This method is
different
```

```
 from the remove() method, because the remove() method will raise an error if the
 specified item does not exist, and the discard() method will not
"""

s={10,30,20,40}
s.discard(30)
print(s)

s.remove(20)
print(s)

s.clear()
print(s)   #make set empty

s.add(50)
print(s)

del s  # delete the set

print()

#set operation on two set
#in operator is is faster in set than list because set uses hasing interll
print("********** set of operation on two set ***********")
s1={2,4,6,8}
s2={3,6,9}

print('union', s1 | s2)
print(s1.union(s2))

print('intersectoin',s1&s2)
print(s1.intersection(s2))

print("present in s1 but not present in s2", s1-s2)
print(s1.difference(s2))

print("symmetric difference, not present in both",s1^s2)
print(s1.symmetric_difference(s2))

print()

#set opeation on two sets
print("********** set operation on two set **********")
s1={2,4,6,8}
s2={4,8}
print("disjoint sets:",s1.isdisjoint(s2))
```

```
print("isSubset:",s1<=s2)
print(s1.issubset(s2))

print("proper set: ",s1<s2)

print("s1 is superset of s2:",s1>=s2)
print(s1.issuperset(s2))

print("s1 is proper superset of s2:",s1>s2)
```

OUTPUT :

********** Creation **********

{10, 20, 30}

{40, 20, 30}

expected type set <class 'dict'>

<class 'set'>

set()

********** Insertion **********

{10, 20, 30}

{10, 20, 30}

{40, 10, 50, 20, 30}

{70, 40, 10, 80, 50, 20, 90, 60, 30}

********** Remove **********

{40, 10, 20}

{40, 10}

set()

{50}

********** set of operation on two set ***********

union {2, 3, 4, 6, 8, 9}

{2, 3, 4, 6, 8, 9}

intersectoin {6}

{6}

present in s1 but not present in s2 {8, 2, 4}

{8, 2, 4}

symmetric difference, not present in both {2, 3, 4, 8, 9}

{2, 3, 4, 8, 9}

********** set operation on two set **********

disjoint sets: False

isSubset: False

False

proper set: False

s1 is superset of s2: True

True

s1 is proper superset of s2: True

4] Dictionary In Python :

```
"""Dictionary is a built-in Python Data Structure that is mutable.
 It is similar in spirit to List, Set, and Tuples."""

#creation
print("********** creation **********")
d={110:"abc",101:"xyz", 105:"pqr"}
print(d)

d={}
d["laptop"]=40000
d["mobile"]=15000
d["earphone"]=1000
print(d)
print(d["mobile"])

#accessing
print("********** Accessing **********")
d={110:"abc",101:"xyz", 105:"pqr"}

print(d.get(101))
print(d.get(125))
print(d.get(125,"NA"))

if 125 in d:
    print(d[125])
else:
    print("NA")

#removal
print("********** Removal **********")
d={110:"abc", 101:"xyz", 105:"pqr", 106:"bcd"}
d[101]="wxy"

print(len(d))
print(d)

print("returning and removing 105 ", d.pop(105))

print("After removing 105 ",d)

del d[106]

print(d)
```

```
d[108]="cde"

print("returning and removing last inserted",d.popitem())
```

OUTPUT :

********** creation **********

{110: 'abc', 101: 'xyz', 105: 'pqr'}

{'laptop': 40000, 'mobile': 15000, 'earphone': 1000}

15000

********** Accessing **********

xyz

None

NA

NA

********** Removal **********

4

{110: 'abc', 101: 'wxy', 105: 'pqr', 106: 'bcd'}

returning and removing 105 pqr

After removing 105 {110: 'abc', 101: 'wxy', 106: 'bcd'}

{110: 'abc', 101: 'wxy'}

returning and removing last inserted (108, 'cde')

5] Count Distinct Element in List :

```
def cDistinct(l):
    res=1

    for i in range(1,len(l)):
        if l[i] not in l[0:i]:
            res=res+1

    return res

l=[10,20,10,30,30,20]

print(cDistinct(l))


print("**************************")

def cDistinct2(l):
    return len(set(l))

print(cDistinct2(l))
```

OUTPUT :

3

****

3

6] subarray with sum zero naïve :

```
def isZeroSum(l):
    n=len(l)

    for i in range(n):
        for j in range(i+1,n+1):
            if sum(l[i:j])==0:
                return True
    return False

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

print(isZeroSum(l))
```

OUTPUT :

True

7] subarray with sum zero Efficient :

```
def isZeroSum(l):
    pre_sum=0

    h=set()

    for i in range(len(l)):
        pre_sum+=l[i]
        if pre_sum==0 or pre_sum in h:
            return True
        h.add(pre_sum)
    return False

l=[4,3,-2,1,1]
print(isZeroSum(l))
```

OUTPUT :

True

8] Check Palindrome Permutation :

```
def isPal(s):
    a=set()
    for i in s:
        if i in a:
            a.remove(i)
        else:
            if(i!="\n") or (i!=" "):
                a.add(i)
    if len(a)<=1:
        return True
    return False

s=input("Enter String: ")
print(isPal(s))
```

OUTPUT 1 :

Enter String: ganesh

False

OUTPUT 2:

Enter String: geeg

True