In [4]:
```
# Function to print all combinations of integers(for 2 elements in a combination)
# [1,2,3] --> (1,2),(1,3),(2,3) --> 3c2 --> 3!/(3-2)!*2!
# Combinations of elements --> ncr --> n!/(n-r)!*r!
# [1,2,3,4] --> (1,2),(1,3),(1,4),(2,3),(2,4),(3,4)


def combinations(li):
    for i in range(len(li)-1):
        for j in range(i+1,len(li)):
            print(li[i],li[j])
combinations([1,2,3,4])
```

```
1 2
1 3
1 4
2 3
2 4
3 4
```

In [12]:
```
# Function to print all combinations of integers(for 3 elements in a combination)
# [1,2,3,4] --> (1,2,3),(1,2,4),(1,3,4),(2,3,4),

def combinations2(li):
    for i in range(len(li)-2):
        for j in range(i+1,len(li)-1):
            for k in range(j+1,len(li)):
                print(li[i],li[j],li[k])
combinations2([1,2,3,4,5])
```

```
1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5
```

In [2]:
```
# Function to identify differences of all pairs of numbers and add those differe
# It returns the updated list and original list
# [1,2,3] -> [1,2,3]

def medium(li,k):
    while(True):
        li3 = differencePairs(li)
        if li3[0] == li3[1]:
            break
    if len(li3[0]) >= k:
        return sorted(li3[0],reverse=True)[k-1]
    return -1



def differencePairs(li):
    cli = li[:]
    newelements = []

    for i in range(len(li)-1):
        for j in range(i+1, len(li)):
            d = abs(int(li[i])-int(li[j]))
            if d not in li and d not in newelements:
                newelements.append(str(d))
    li.extend(newelements)
    return [cli, li]

li = [3,6,24]
max(li)
```

Out[2]: 24

In [52]:
```
# Find the difference of elements in a list and insert them inthe list, then find

def difference(li,k):
    #newelements=[]
    for i in li:
        for j in li:
            d=abs(i-j)
            if d!=0:
                if d not in li:
                    li.append(d)

    li=sorted(li,reverse=True)
    print(li)
    unique=[]
    for i in li:
        if i not in unique:
            unique.append(i)
    print(unique)
    if len(unique)>=k:
        return unique[k-1]
    return -1

difference([1,3,5,7,7],3)
```

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

Out[52]: 5

In [44]:
```
# List Data Referencing vs DataCopy

a=[1,2,3]
b=[1,3,2]
a=b.copy   # Data Copy though indirect referencing
a=b[:]     # Data Copy through direct referencing
print(a)
print(b)
```

```
[1, 3, 2]
[1, 3, 2]
```

In [ ]:
```

```

Set-Data Structure in Python

- Set represented by '{ }'
- It contains only unique elements
- It is Mutable(changable)

In [4]:
```
a={1,2,3,4,5}   # Example of set

a={1,2,3,4,5,5}  #If we are same elements in a set, then it takes only one time

a.add(6)   # Adding the element to the set
a
```

Out[4]: {1, 2, 3, 4, 5, 6}

In [7]:
```
# Accessing the elements in the set

for i in a:
    print(i,end=' ')
```

```
1 2 3 4 5 6
```

In [7]:
```
# Adding the multiple elements in the list

b={7,8,9,0}
a.update(b)
a
```

Out[7]: {0, 1, 2, 3, 4, 5, 7, 8, 9}

In [6]:
```
# Set removes the duplicate values

c={1,3,5,7,8}
a.update(c)
a
```

Out[6]: {0, 1, 2, 3, 4, 5, 7, 8, 9}

In [8]:
```
# Adding the multiple elements

c={7,8,9,1,2,3}
li=[11,12,13]
c.update(c,li)
c
```

Out[8]: {1, 2, 3, 7, 8, 9, 11, 12, 13}

In [10]:
```
# Removing the elements in the set

a.discard(1)
a
```

Out[10]: {0, 2, 3, 4, 5, 7, 8, 9}

In [17]:
```
# All elements in both lists without repeating
# AUB = BUA

a.union(b)
b.union(a)

# Common elements in a set
a.intersection(b)

# Check the Common elements in two sets then return True/False
a.isdisjoint(b)
```

Out[17]:
```
False
```

In [28]:
```
# All elements of a which are not in b
# Ex:a={10,1,2,3,4,5,6} and b={1,2,3,7,8} --> {4,5,6,10}
a-b

# All elements of a which are not in b
b-a
```

Out[28]:
```
{2, 3, 4, 5}
```

In [21]:
```
# Elements either in a or in b i.e, uncommon elements

a^b
```

Out[21]:
```
{2, 3, 4, 5}
```

In [23]:
```
# Sorting the elments

sorted(a)
```

Out[23]:
```
[0, 2, 3, 4, 5, 7, 8, 9]
```

In [26]:
```
# Creating the empty set

d=set()
d
```

Out[26]:
```
set()
```

In [27]:
```
# Deleting Duplicate elements and printing the original list

li=[1,2,3,4,5,2,3,1,4,2]
u=set(li)
print(u)
```

```
{1, 2, 3, 4, 5}
```

In [ ]:
```

```

- Procedural : C

- Object Oriented : Java, Python
- Scripting : PHP, Javascript, Shell, Perl
- Functional : Python, Haskell, Scala
- Logic(Rule) : Prolog, Lisp

List Cmprehensions

In [13]:
```
# Print the N natural numbers in a list

n=10
li=[]
for i in range(1,n+1):
    li.append(i)
print(li)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [19]:
```
# Print the N natural numbers in a list using List Comprehension

li=[i for i in range(1,11)]
print(li)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [21]:
```
# Apply list comprehension to  store the cubes on N natural numbers

li=[i**3 for i in range(1,11)]
li
```

Out[21]:
```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

In [22]:
```
# Function to calculate the factorial(using recursion)

def factorial(n):
    if n==0 or n==1:
        return 1
    return n*factorial(n-1)


# Apply list comprehension to calculate factorial of N natural numbers

n=5
factorialList=[factorial(i) for i in range(1,n+1)]
factorialList
```

Out[22]:
```
[1, 2, 6, 24, 120]
```

In [37]:
```
# Store the cummulative sum of numbers till the n.
# n=3 --> [1,1+2,1+2+3] --> [1,3,6]

cummulativeSum=[sum(range(1,i+1)) for i in range(1,n+1)]
cummulativeSum
```

Out[37]:
```
[1, 3, 6, 10, 15]
```

In [42]:
```
# List Comprehension to store only leap years in a given range

st=1970
et=2019
leapYears=[i for i in range(st,et+1)
           if ((i%400==0) or (i%100!=0 and i%4==0))]
leapYears
```

Out[42]:
```
[1972, 1976, 1980, 1984, 1988, 1992, 1996, 2000, 2004, 2008, 2012, 2016]
```

In [30]:
```
# Extract the unique data from given list by using List Comprehenssion

li=[1,2,3,2,1]
li.sort()
u2=[]
[u2.append(i) for i in li if i not in u2 ]
u2
```

Out[30]:
```
[1, 2, 3]
```

In [ ]:
```

```

Iterators

- Iterable
- Ex: Strings, Lists, Tuples, Sets, Dictionaries
- Convert Iterable to Iterator --> iter()
- Difference of For Loop and Iterator:
  - For Loop:It can stop only execution is completed or after break condition
  - Iterator: You can stop iteration any where in the loop

In [54]:
```
it=iter('Python')

print('1: ')
print(next(it))
print('\n')
print('2: ')
print(next(it))
print('\n')
print('3: ')
print(next(it))
print('\n')
print('4: ')
print(next(it))
print('\n')
```

```
1:
P


2:
y


3:
t


4:
h
```

Generator

- Generator is a function

In [65]:
```
# Using Generator print the cube of number

def generator():
    n=2
    yield n

    n=n**3
    yield n

    n=n**3
    yield n

a=generator()
next(a)
next(a)
next(a)
# Here Generator runs based on no.of yield condition
```

Out[65]:
```
512
```

In [72]:
```
def generator():
    n=2
    for i in range(1,5):
        n**=3
        yield n
a=generator()
next(a)
next(a)
next(a)
next(a)
# Here Generator runs based on for loop range
```

Out[72]:
```
241785163922925834941235 2
```

In [75]:
```
def generator():
    n=2
    while True:
        n**=3
        yield n
a=generator()
next(a)
next(a)
next(a)
next(a)
# Here Generator runs how many times you want because of while loop
```

Out[75]:
```
1413477651822707463666638000594334812661987117500495166497284961034095820 8
```

In [76]:
```
def generator():
    n=2
    while True:
        n**=3
        yield n
a=generator()
for i in range(5):
    print(next(a))
# Here while loop waited, so we want to print next number also
```

```
8
512
134217728
241785163922925834941235 2
1413477651822707463666638000594334812661987117500495166497284961034095820 8
```

In [ ]:
```

```