Day Objectives:-

- Combinations
- Set-Data Structure
- Set Operations
- Use cases
- Functions Programming
- Iterators
- Generators
- Maps
- Lambda
- Builtin Functions
- Use cases

In [37]:

```
#Function to print all combinations of pairs of integers in a unique list
#[1,2,3]---->(1,2),(1,3),(2,3)  3C2---> (3!/((3-2)!*2!))

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

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

In [7]:

```python
#Function to print all combinations of pairs of integers in a unique list
#[1,2,3]---->(1,2),(1,3),(2,3)  3C2---> (3!/((3-2)!*2!))

def combinations3(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])
    return
li=[1,2,3,4,5]
combinations3(li)
```

```
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 [23]:

```
# Function to find the klargestdifferencepairs

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

    # Function to identify differences of all pairs of numbers
    # Pairs of numbers and add those differences to the same list
    # It returns the updated list and original list

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(d)
    li.extend(newelements)
    return [cli,li]
li=[2,3,6,9,12,1,4,7,10,5,8,11]
k=int(input())
DifferencePairs(li)
```

```
2
```

Out[23]:

```
11
```

In [ ]:

```
# [4,8]
[20,40,60]
[4,8,12,16]
[3,6,9,12]
#Convert the list into an arithmetic progression
[3,8,15,5,2,1,4,6,7,9,10,11,12,13,14]

a=[1,2,3]
b=[1,3,2]
a=b.copy() # data accessing by indirect refrence
a=b[:] # direct accessing
```

Type *Markdown* and LaTeX: α^2

Min- Max hacker earth problem

In [27]:

```
n1=int(input())
n=list(map(int,input().split()))
n3=max(n)
n2=min(n)
c=0
for i in range (n2,n3+1):
    if i in n:
        c=c+1
if c==n1:
    print("YES")
else:
    print("NO")
```

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

Min- Max hacker earth problem model2

In [26]:

```
n1=input().split()
x=int(n1[1])
y=int(n1[2])
n=list(map(int,input().split()))
c=0
for i in range (x,y+1):
    if i in n:
        c=c+1
if c==n:
    print("YES")
else:
    print("No")
```

```
5 1 5
1 2 3 4 5
No
```

Set-Data Structure in Python

- Represented by '{}'
- Sets are mutable
- There is no order for this sets # example a[1] it gives type error

In [45]:

```
a={1,2,3,4,5,6,6}
# Set contains only unique elements no repetations
a.add(7) # Adding a single element to the set
a
```

Out[45]:

```
{1, 2, 3, 4, 5, 6, 7}
```

```
In [46]: a

Out[46]: {1, 2, 3, 4, 5, 6, 7}
```

```
In [47]: for i in a:      # Accessing elements in a set
             print(i,end=" ")

         1 2 3 4 5 6 7
```

```
In [48]: b={8,6,7,7,3,4,1,2,3}
         a.update(b)
         b
         a

Out[48]: {1, 2, 3, 4, 5, 6, 7, 8}
```

```
In [50]: b={7,8,9,1,2,3}
         li=[11,12,13]
         a1={3,8,9}
         a1.update(b,li)

Out[50]: {1, 2, 3, 7, 8, 9, 11, 12, 13}
```

```
In [51]: a1.discard(13)  # removing the element
```

```
In [52]: a1

Out[52]: {1, 2, 3, 7, 8, 9, 11, 12}
```

```
In [55]: a1.remove(7)
         a1

Out[55]: {1, 2, 3, 8, 9}
```

```
In [62]: a1.remove(1)
         a1

Out[62]: set()
```

```
In [75]: a={10,1,2,3,4,5,6}
         b={7,8,9,1,2,3}
         c={111,123}
         a.intersection(b)

Out[75]: {1, 2, 3}
```

```
In [77]: b

Out[77]: {1, 2, 3, 7, 8, 9}
```

```
In [78]: a
```

```
Out[78]: {1, 2, 3, 4, 5, 6, 10}
```

```
In [79]: a.union(b)  # A U B
```

```
Out[79]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
In [80]: b.union(a) # B U A
```

```
Out[80]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
In [81]: a.intersection(b) # A intersection with b
```

```
Out[81]: {1, 2, 3}
```

```
In [82]: b.intersection(a) # B intersection with a
```

```
Out[82]: {1, 2, 3}
```

```
In [83]: a.isdisjoint(b) # disjoint set
```

```
Out[83]: False
```

```
In [84]: a-b # All elements of a which are not in b
```

```
Out[84]: {4, 5, 6, 10}
```

```
In [85]: b-a # All elements which are not in a
```

```
Out[85]: {7, 8, 9}
```

```
In [88]: a1=sorted(a)  # By using the sorting we operate the slice operations in set with
         a1
```

```
Out[88]: [1, 2, 3, 4, 5, 6, 10]
```

```
In [89]: a1[1]
```

```
Out[89]: 2
```

```
In [92]: a^b # Elements either in a or b (uncommon elements)
```

```
Out[92]: {4, 5, 6, 7, 8, 9, 10}
```

```
In [96]: d=set()
         d
```

```
Out[96]: set()
```

In [97]:
```
li=[1,2,3,4,5,6,1,2,3,4]
u=set(li)
u
```

Out[97]:
```
{1, 2, 3, 4, 5, 6}
```

In [ ]:

Type *Markdown* and LaTeX: α^2

In [ ]:
```
- Procedural : C

- Object Oriendted : Java

- Scripting : PHP,Python,Javascript,Shell,Perl

- Functional : Python (Python is a Scripting language ),Haskell,Scala

- Logic : logic(means Rules) Prolog,Lisp
```

List Comprehensions

In [102]:
```
# List of N Natural Numbers
n=int(input())
l=[]
for i in range(1,n+1):
    l.append(i)
print(l)
```

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

In [108]:
```
li=[i for i in range(1,11)]
print(li)
```

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

In [111]:
```
# Apply list comprehension to store the cubes of n natural numbers

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

Out[111]:
```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

In [112]:

```
# Function  to calculate the factorial
def factorial (n):
    if n==0 or n==1:
        return 1
    else:
        return n*factorial(n-1)
n=int(input())
factorial(n)
```

Out[112]:

```
120
```

In [121]:

```
# Apply list comprehension to calculate the factorial of n
factorialList=[factorial(i) for i in range(1,n+1)]
n=int(input())
factorialList
```

```
12
```

Out[121]:

```
[1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800]
```

In [136]:

```
# Store cumulative sum of numbers till n in a list
# n=5----->[1,3,6,10,15]

def cum(n):
    sum=0
    for i in range(1,n+1):
        sum=sum+i
        print(sum)
   #return sum
n=int(input())
cum(n)
```

```
4
1
3
6
10
```

In [137]:

```
N=int(input())
cumlist=[sum(range(1,i+1)) for i in range(1,N+1)]
cumlist
```

```
5
```

Out[137]:

```
[1, 3, 6, 10, 15]
```

In [ ]:

```
# List comprehension to store only leap years in a given time period

st=1970
et=2019
leapYears=[1972,1976,1980,....2016]
```

In [143]:

```
def isleap(y):
    if(y%4==0 or y%100!=0 and y%400==0):
        return True
    else:
        return False
#y=int(input())
isleap(y)

def rangeyear(st,et):
    li=[]
    for i in range(st,et+1):
        if isleap(i):
            li.append(i)
    return li
st=int(input())
et=int(input())
rangeyear(st,et)
```

```
1970
2019
```

Out[143]:

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

In [148]:

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

```
1970
2019
```

Out[148]:

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

In [155]:

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

Out[155]:

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

In [1]:

```
li=[1,2,3,2,1]
li.sort()
unique=[]
unique=[li[i] for i in range(0,len(li)-1) if li[i]!=(li)[i+1]]
unique
```

Out[1]:

```
[1, 2]
```

Iterators

- Iterable -String,Lists,Tuples,Sets,Dictionaries
- Convert iterable to iterator----> iter()
- for loop : We can not break until some condition is reached
- Iterator : We can stop at anytime (There is a pause in iterable process)

In [4]:

```
it =iter('Python')
print('1:')
print(next(it))
print('\n')
print('2:')
print(next(it))
```

```
1:
P


2:
y
```

Generators

- Generator is a user defined function
- Yield is like a return

In [9]:

```
def generator():
    n=2
    for i in range(1,5):
        n**=3
        yield n
a=generator()
next(a)
next(a)
```

Out[9]:

```
512
```

In [13]:

```
# for infinite Loop
def generator():
    n=2
    while True:
        n**=3
        yield n
a= generator()
next(a)
#b=next(a)**2
#b*=next(a)
#b
for i in range(4):
    print(next(a))
```

```
512
134217728
2417851639229258349412352
14134776518227074636666380005943348126619871175004951664972849610340958208
```

In [ ]: