

Combinations

```
In [6]: 1 # Function to print all combinations of pairs of intergers in a unique list
2 # [1,2,3] --> (1,2),(1,3),(2,3) -> 3C2 --> 3!/((3-2)!*2!) --> NCR
3
4 #[1,2,3,4] --> two elements (1,2),(1,3),(1,4),(2,3),(2,4),(3,4) --> tuples
5
6 def Combinations(li):
7     for i in range(len(li)-1):
8         for j in range(i+1,len(li)):
9             print(li[i],li[j])
10 li = [1,2,3,4]
11 Combinations(li)
```

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

```
In [11]: 1 # Function to print all combinations of pairs of intergers in a unique list
2
3 #[1,2,3,4] --> (1,2,3),(1,2,4),(1,3,4),(2,3,4)
4
5 def Combinations(li):
6     for i in range(len(li)-2):
7         for j in range(i+1,len(li)-1):
8             for k in range(j+1,len(li)):
9                 print(li[i],li[j],li[k])
10 li = [1,2,3,4,5]
11 Combinations(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 [ ]: 1 def medium(li,k):
        2     while(True):
        3         li3 = differencePairs(li)
        4         if li3[0]==li3[1]:
        5             break
        6         if(len(li3[0]))>=k:
        7             print(li)
        8             return sorted(li3[0],reverse=True)[k-1]
        9         else:
        10            return -1
        11
        12 # Function to identify differences of all
        13 # pairs of numbers and add those differences
        14 # to same list.
        15 # this function returns updated list and original list
        16 def differencePairs(li):
        17     cli=li[:] # Coping the li values into cli
        18     newelements = []
        19
        20     for i in range(len(li)-1):
        21         for j in range(i+1,len(li)):
        22             d = abs(int(li[i])-int(li[j]))
        23             if d not in li and d not in newelements:
        24                 newelements.append(str(d))
        25     li.extend(newelements)
        26     return [cli,li]
        27
        28 with open('DataFiles/medium-input.txt','r') as f:
        29     t=int(f.readline())
        30     for i in range(t):
        31         n=f.readline()
        32         li=f.readline().split()
        33         k=int(f.readline())
        34         medium(li,k)
        35
        36 #li = [3,6,9]
        37 #differencePairs(li)
        38
        39

```

```

In [45]: 1 # List Data referencing vs Data copy
        2 a = [1,2,3]
        3 b= [1,3,2]
        4 a = b.copy() # Data Copy through indirect referencing
        5 a=b[:] # Data copy through direct referencing
        6
        7 a = b # Data Referencing
        8
        9 b.append(4)
       10 a.append(5)
       11 b

```

Out[45]: [1, 3, 2, 4, 5]

```
In [25]: 1 r=[1,2,3,4,5,6]
          2 s=[7,8]
          3 r.extend(s)
          4 r
```

```
Out[25]: [1, 2, 3, 4, 5, 6, 7, 8]
```

```
In [40]: 1 # Below lists are the examples of constant list
          2 # When we do the difference we will get same elements in the list
          3 # Don't need to do any append
          4
          5 [4,8]
          6 [20,40,60]
          7 [4,8,12,16]
          8 [3,6,9,12]
          9 # Convert the list into an
         10 # Arithmetic Progression (differences are same here in those lists)
         11
         12 [3,8,15]
         13 [3,8,15,5,2,1,4,6,7,9,10,11,12,13,14]
```

```
Out[40]: [4, 8]
```

```
In [ ]: 1 def medium(li,k):
          2     while(True):
          3         li3 = differencePairs(li)
          4         if li3[0] == li3[1]:
          5             break
          6         if(len(li3[0]))>=k:
          7             return sorted(li3[0],reverse=True)[k-1]
          8         else:
          9             return -1
         10 def differencePairs(li):
         11     c = li.copy()
         12     newelements = []
         13     for i in range(len(li)-1):
         14         for j in range(i+1,len(li)):
         15             d = abs(li[i]-li[j])
         16             if d not in li and d not in newelements:
         17                 newelements.append(d)
         18     li.extend(newelements)
         19     return li
         20
         21 li = [2,3,6,9,12,1,4,7,10,5,8,11]
         22 differencePairs(li)
         23 medium(li,2)
```

Set - Data Structure in python

- Represented by {}
- it contains only unique data repeated elements will remove automatically
- Set mutable
- index operations will not work here in set

```
In [4]: 1 a = {1,2,3,4,5,6,6,5}
        2
        3 a.add(7)    # Adding a single element to set
        4 a
```

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

```
In [5]: 1 for i in a:
        2     print(i,end=' ')    #--> Accessing elements in a set
        3
        1 2 3 4 5 6 7
```

```
In [9]: 1 # adding
        2 b = {7,8,9,1,2,3}    # Remove duplicate elements adds only unique elements
        3 li=[22,33]
        4 a.update(b)
        5 a
```

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

```
In [13]: 1 li = [22,44]
        2 a.update(li)
        3 a
```

Out[13]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 22, 44}

```
In [15]: 1 s='sirei'
        2 a.update(s)
        3 a
```

Out[15]: {1, 2, 22, 3, 4, 44, 5, 6, 7, 8, 9, 'e', 'i', 'r', 's'}

```
In [16]: 1 p=[334,488483,393]
        2 st = 'upma'
        3 a.update(b,p)
        4 a
```

Out[16]: {1, 2, 22, 3, 334, 393, 4, 44, 488483, 5, 6, 7, 8, 9, 'e', 'i', 'r', 's'}

```
In [18]: 1 # To remove a element i.e only a single element can be removed by discard f
        2 a.discard(44)
        3 a
```

Out[18]: {1, 2, 22, 3, 334, 393, 4, 488483, 5, 6, 7, 8, 9, 'e', 'i', 'r', 's'}

```
In [23]: 1 a = { 10,1,2,3,4,5,6}
          2 b = {7,8,9,1,2,3}
          3
          4 a.union(b)
          5
          6 # A U B = B U A    ---> a union b is equal to b union a
          7
```

Out[23]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

```
In [24]: 1 b.union(a)
```

Out[24]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

```
In [25]: 1 # Accessing common elememnts in both sets
          2
          3 a.intersection(b)
```

Out[25]: {1, 2, 3}

```
In [26]: 1 # If both sets are not have common elements then it will give o/p as TRUE ot
          2 c = {111,123}
          3 a.isdisjoint(b)
```

Out[26]: False

```
In [28]: 1 c = {111,123}
          2 a.isdisjoint(c)
```

Out[28]: True

```
In [29]: 1 a - b    # ALL elements in a which are not in b
```

Out[29]: {4, 5, 6, 10}

```
In [30]: 1 b - a    # Return all elements in b which are not in a
```

Out[30]: {7, 8, 9}

```
In [31]: 1 a
```

Out[31]: {1, 2, 3, 4, 5, 6, 10}

```
In [32]: 1 b
```

Out[32]: {1, 2, 3, 7, 8, 9}

```
In [33]: 1 sorted(a)
```

Out[33]: [1, 2, 3, 4, 5, 6, 10]

```
In [34]: 1 a^b  # Elements either in a or in b i.e not common i.e not intersection
          2
```

```
Out[34]: {4, 5, 6, 7, 8, 9, 10}
```

```
In [35]: 1 t = { 1,4,33,393,292,8,29,9}
          2 sorted(t)
```

```
Out[35]: [1, 4, 8, 9, 29, 33, 292, 393]
```

```
In [36]: 1 # Creating an empty set
          2 d = set()
          3 d
```

```
Out[36]: set()
```

```
In [38]: 1 li = [32, 34,22,1,2,3,4,1,2,4,56,]
          2 u = set(li)
          3 u
```

```
Out[38]: {1, 2, 3, 4, 22, 32, 34, 56}
```

Functional Programming

- C Language is procedural or structural programming
- C++,Java are Object oriented languages
- Python,JavaScript,PHP are Scripting Languages

Procedural : C

Object Oriented : Java, Python

Scripting : PHP, Python, Javascripting, Shell, Perl

Functional : Python, Haskell, Scala

Logic : Prolog, Lisp

Functional Programming

- **Def** : In computer science, functional programming is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
- Search in google as "**Functionl Programming**"

List Comprehensions

- mathematical or symbolic logic into a list

```
In [41]: 1 # N natural numbers in a list
        2 n = 10
        3 li = []
        4 for i in range(1,n+1):
        5     li.append(i)
        6 print(li)
        7
```

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

For above program we need 5 lines by using List Comprehension we get in single line

```
In [42]: 1 li = [i for i in range(1,11)]
        2 li
```

Out[42]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

```
In [43]: 1 # Apply list comprehension to store the
        2 # Cubes on n natural numbers
        3
        4 li = [i**3 for i in range(1,11)]
        5 li
```

Out[43]: [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]

```
In [48]: 1 # Function to calculate the factorial
        2 def factorial(n):
        3     if n == 0 or n == 1:
        4         return 1
        5     return n * factorial(n-1)
        6 # factorial(5)
```

```
In [49]: 1 # Apply list comprehension to calculate
        2 # Factorial of n natural numbers
        3
        4 n = 10
        5 factorialList = [factorial(i) for i in range(1,n+1)]
        6 factorialList
```

Out[49]: [1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800]

```
In [63]: 1 # Store cumulative sum of numbers till n in a list
2 # n = 5 --> [1, 3, 6, 10, 15]
3
4 def cummsum(n):
5     li = []
6     for i in range(1,n+1):
7         li.append(i)
8     #return li
9     s=0
10    for i in li:
11        sum(li[:i+1])
12    return s
13
14 cummsum(5)
15
```

File "<ipython-input-63-c0d7d9b4c166>", line 11
 s=sum(li[:i+1])
 ^

SyntaxError: invalid syntax

```
In [62]: 1 li = [1,2,3,4,5,6]
2 sum(li[0:4]) # --> ouput upto 4
```

Out[62]: 10

```
In [65]: 1 n = 5
2 cumulativeSum = [sum([i]) for i in range(1,n+1)]
3 cumulativeSum
```

Out[65]: [1, 2, 3, 4, 5]

```
In [73]: 1 cumulativeSum = [sum(range(1,i+1)) for i in range(1,n+1)]
2 cumulativeSum
```

Out[73]: [1, 3, 6, 10, 15]

```
In [78]: 1 # List Comprehensions to store
2 # Only Leap Years in a given time period
3
4 st = 1970
5 et = 2019
6 leapYears = [i for i in range(st,et+1)
7               if i%400==0 or (i%100!=0 and i%4 == 0) ]
8 leapYears
```

Out[78]: [1972, 1976, 1980, 1984, 1988, 1992, 1996, 2000, 2004, 2008, 2012, 2016]


```
In [81]: 1 # If we have a if condition then it is after for loop condition
2 st = 1970
3 et = 2019
4 leapYears = [[(i%400==0 or (i%100!=0 and i%4 == 0)),i] for i in range(st,et+1)]
5 leapYears
```

```
Out[81]: [[False, 1970],
[False, 1971],
[True, 1972],
[False, 1973],
[False, 1974],
[False, 1975],
[True, 1976],
[False, 1977],
[False, 1978],
[False, 1979],
[True, 1980],
[False, 1981],
[False, 1982],
[False, 1983],
[True, 1984],
[False, 1985],
[False, 1986],
[False, 1987],
[True, 1988],
[False, 1989],
[False, 1990],
[False, 1991],
[True, 1992],
[False, 1993],
[False, 1994],
[False, 1995],
[True, 1996],
[False, 1997],
[False, 1998],
[False, 1999],
[True, 2000],
[False, 2001],
[False, 2002],
[False, 2003],
[True, 2004],
[False, 2005],
[False, 2006],
[False, 2007],
[True, 2008],
[False, 2009],
[False, 2010],
[False, 2011],
[True, 2012],
[False, 2013],
[False, 2014],
[False, 2015],
[True, 2016],
[False, 2017],
[False, 2018],
[False, 2019]]
```

```
In [91]: 1 li = [1,2,3,2,1]
        2 unique = [i for i in range(0,len(li)) if sorted(li.count(i) == 1]
        3 unique
```

Out[91]: [3]

Iterators

- **Iterable** - Strings, Lists, Tuples, Sets, Dictionaries
- Convert Iterable to Iterator -> iter()
- **For Loop** : goes through every element untill condition reached
- **Iterator**: we can stop at anywhere and start at any time

```
In [96]: 1 it = iter('Python')
        2 next(it)    # --> 'P'
        3 next(it)    # --> 'y'
```

Out[96]: 'y'

```
In [97]: 1 it = iter('Python')
        2 print(next(it))
        3 print(next(it))
```

P
y

```
In [105]: 1 it = iter('Python')
        2
        3 for i in it:
        4     print(next(it))
```

y
h
n

```
In [106]: 1 it = iter('Python')
        2
        3 print('1: ')
        4 print(next(it))
        5 print('\n')
        6 print('2: ')
        7 print(next(it))
```

1:
P

2:
y

Generators

- Generator is Function

In [117]:

```
1
2 def generator():
3     n = 2
4     yield n
5     n = n ** 3
6     yield n
7
8     n = n ** 3
9     yield n
10 a = generator()
11 print(next(a))
12 print(next(a))
13 print(next(a))
14
```

```
2
8
512
```

In [120]:

```
1 def generator1():
2     n = 2
3     for i in range(1,5):
4         n **= 3
5         yield n
6 a = generator1()
7
8 print(next(a))
9 print(next(a))
```

```
8
512
```

```
In [127]: 1 def generator2():
2         n = 2
3         while(True):
4             n **= 3
5             yield n
6
7         a = generator2()
8
9         # print(next(a))
10        # b=next(a)**2
11        # b *= next(a)
12        # print(next(a))
13
14        for i in range()
15
16
17
```

8

2417851639229258349412352

In []:

1