Day Objectives:

- Functional programming in python.
- Python packages.
    - Numpy
    - Pandas
    - MatPlotLib

Functional programming in python

- List comprehension
- Maps
- Lambda
- Filters
- Reduce

Functional programming

- It reduces the number of lines in code.
- It is more efficient than compared to other programming languages.

List comprehension

- It is used to create a list in simple way compared to loops.
- It is more efficient than loops.

In [1]:

```
for i in range(1,11):
    print(i,end=" ")
```

```
1 2 3 4 5 6 7 8 9 10
```

In [2]:

```
s=input()
import re
if s in re.findall("^[a-z]{5}[!@#$%^]{6}$",s):
    print(s)
else:
    print("choose a correct one")
```

```
kshfd
choose a correct one
```

In [3]:

```
l2=[i**2 for i in range(1,11)]
l2
```

Out[3]:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [4]:

```
def factorial(n):
    if n==0 or n==1:
        return 1
    return n*factorial(n-1)
#factorial(5)
#by using list comprehension
lch=[factorial(i) for i in range(1,11)]
lch
```

Out[4]:

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

In [5]:

```
def cumulative(n):
    t=0
    for i in range(1,n+1):
        t=t+i
    return t
cumulative(5)
```

Out[5]:

```
15
```

In [6]:

```
#Task-1:
#cumulative sum of elements using list comprehension
n=int(input())
def cumulative(s):
    t=0
    for i in range(1,s+1):
        t=t+i
    return t
l=[cumulative(n) for n in range(1,n+1)]
print(l)
```

```
5
[1, 3, 6, 10, 15]
```

In [13]:

```
#Task-2:
#Print leap years in a given range
s=int(input())
def leap(n):
    for i in range(1,n):
        if(n%100!=0 and n%4==0) or (n%400==0):
            print(n,"is leap year")
            n+=1
l=[leap(s) for s in range(1,s+1)]
l
```

```
2019
4 is leap year
8 is leap year
12 is leap year
16 is leap year
20 is leap year
24 is leap year
28 is leap year
32 is leap year
36 is leap year
40 is leap year
44 is leap year
48 is leap year
52 is leap year
56 is leap year
60 is leap year
64 is leap year
68 is leap year
72 is leap year
76 is leap year
```

In [9]:

```
l=[n for n in range(1990,2019) if n%400==0 or (n%4==0 and n%100!=0)]
l
```

Out[9]:

```
[1992, 1996, 2000, 2004, 2008, 2012, 2016]
```

In [10]:

```
l=[s for s in range(1,10) if s%2==0]
l
```

Out[10]:

```
[2, 4, 6, 8]
```

In [11]:

```
l=[i for i in range(1,11) if i%2!=0]
l
```

Out[11]:

```
[1, 3, 5, 7, 9]
```

Python special functions

- 1.map()
- 2.filter()
- 3.reduce()
- 4.lambda

In [15]:

```
#Syntax:
  #map(functionname,sequence)
#Find the double of elements in a list
def double(n):
    return n+n
l=[23,45,67,88,40,45]
data=list(map(double,l))
print(data)
```

```
[46, 90, 134, 176, 80, 90]
```

In [16]:

```
def power(n):
    return pow(n,2)
l=[23,45,67,88,40,45]
data=list(map(power,l))
print(data)
```

```
[529, 2025, 4489, 7744, 1600, 2025]
```

In [18]:

```
#map() with multi parameters
def sumof(x,y,z):
    return x+y+z
print(list(map(sumof,[1,2,3],[4,5,6],[7,8,9])))
```

```
[12, 15, 18]
```

In [2]:

```
#Task-3:
def lwrtoupr(s):
    return str(s).upper()
def printiter(it):
    for x in it:
        print(x,end=" ")
    print(" ")
#map() with string
map_iter=map(lwrtoupr,'abc')
print(list(map_iter))
```

```
['A', 'B', 'C']
```

In [1]:

```
#Best answer:
def upper(abc):
    return "A B C"
upper("abc")
```

Out[1]:

```
'A B C'
```

In [4]:

```
#Filtering:
#Syntax:
#filter(functionname,sequence)
#number Filtering
def votereligible(age):
    if age>=18:
        return True
    else:
        return False
l=[23,12,45,21,18,19,5,3,35]
print(list(filter(votereligible,l)))
```

```
[23, 45, 21, 18, 19, 35]
```

In [8]:

```
#String Filtering
def vowelfilter(var):
    vowels=['a','e','i','o','u']
    if var in vowels:
        return True
    else:
        return False
strings=['t','e','j','i','k','g','c','a','b','b','a','g','e']
print(list(filter(vowelfilter,strings)))
strings=['v','i','j','a','y','a','l','a','s','m','i']
print(list(filter(vowelfilter,strings)))
```

```
['e', 'i', 'a', 'a', 'e']
['i', 'a', 'a', 'a', 'i']
```

In [18]:

```
#Task-4:
def numfilter(n):
    if (n%3==0 and n%5==0):
        return True
    else:
        return False
num=[35,58,69,13,12,24,89,445,345,93,27,36]
print(list(filter(numfilter,num)))
```

```
[345]
```

In [20]:

```
#Reduce()
#Syntax:
from functools import reduce
def add(x,y):
    return x+y
print("Sum of 7 is",reduce(add,range(1,8)))
```

```
Sum of 7 is 28
```

In [24]:

```
from functools import reduce
def fact(x,y):
    return x*y
print("Factorial of 7 is",reduce(fact,range(1,8)))
```

```
Factorial of 7 is 5040
```

In [25]:

```
#Lambda
#Syntax:
lambda arguments:expression
```

Out[25]:

```
<function __main__.<lambda>(arguments)>
```

In [29]:

```
#Lambda with map()
print(list(map(lambda x:x**2,[1,2,3,4,5])))
#Lambda with filter()
print(list(filter(lambda x:x%3==0,[3,6,8,95,24])))
#Lambda with reduce()
print(reduce(lambda x,y:x*y,[2,3,4,5,6,7,8,9]))
```

```
[1, 4, 9, 16, 25]
[3, 6, 24]
362880
```

Python libraries for Data Science

Data Science:

- It is the processing of deriving knowledge and insights from a huge and diverse set of data through organizing and analyzing and processing the data.
- There are nearly 20 python libraries for data science but mostly we are using these five:
  - Tensor flow
  - Numpy
  - Pandas
  - MatpLot Library
  - SciPy

Numpy(Numerical python)

- Numpy is mainly used for creation of arrays like 1D,2D,3D.

In [31]:

```
import numpy as np
np.__version__
```

Out[31]:

```
'1.16.5'
```

In [32]:

```
a=np.array([1,2,3,4])
a
```

Out[32]:

```
array([1, 2, 3, 4])
```

In [33]:

```
a=np.array((1,2,3,4))
a
```

Out[33]:

```
array([1, 2, 3, 4])
```

In [34]:

```
a=np.array([[1,2,3,4],[5,6,7,8]])
a
```

Out[34]:

```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

In [36]:

```
a=np.array([[[1,2,3],[4,5,6],[7,8,9,]]])
a
print(a)
print(a.ndim)            #returns the dimensions of array
print(a.shape)           #returns the shape of array
print(a.size)            #returns the number of elements in array
print(a.dtype)           #returns the each element's datatype
print(a.itemsize)        #returns the each element size in bytes
```

```
[[[1 2 3]
  [4 5 6]
  [7 8 9]]]
3
(1, 3, 3)
9
int32
4
```

In [37]:

```
help(np.array)
```

```
Help on built-in function array in module numpy:

array(...)
    array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

    Create an array.

    Parameters
    ----------
    object : array_like
        An array, any object exposing the array interface, an object whose
        __array__ method returns an array, or any (nested) sequence.
    dtype : data-type, optional
        The desired data-type for the array.  If not given, then the type
will
        be determined as the minimum type required to hold the objects in
the
        sequence.  This argument can only be used to 'upcast' the array.
For
```

In [39]:

```
#np.arange()
print(np.arange(10))
print(np.arange(1,100,2))
print(np.arange(30,40))
print(np.arange(3,5,0.1))
print(np.arange(2.5,10,0.75))
```

```
[0 1 2 3 4 5 6 7 8 9]
[ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95
 97 99]
[30 31 32 33 34 35 36 37 38 39]
[3.  3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.  4.1 4.2 4.3 4.4 4.5 4.6 4.7
 4.8 4.9]
[2.5  3.25 4.   4.75 5.5  6.25 7.   7.75 8.5  9.25]
```

In [40]:

```
#linspace()
print(np.linspace(1,10,10))
print(np.linspace(1,100,10))
print(np.linspace(1,100,20))
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[  1.  12.  23.  34.  45.  56.  67.  78.  89. 100.]
[  1.           6.21052632  11.42105263  16.63157895  21.84210526
  27.05263158  32.26315789  37.47368421  42.68421053  47.89473684
  53.10526316  58.31578947  63.52631579  68.73684211  73.94736842
  79.15789474  84.36842105  89.57894737  94.78947368 100.        ]
```

In [41]:

```
#reshape(rows,columns)
b=np.arange(9).reshape(3,3)
print(b)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

In [43]:

```
#zeros()
np.zeros((3,2))
```

Out[43]:

```
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
```

In [44]:

```
#full()
np.full((3,2),5)
```

Out[44]:

```
array([[5, 5],
       [5, 5],
       [5, 5]])
```

In [45]:

```
#eye()
np.eye(3)
```

Out[45]:

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [48]:

```
#diag()
a=np.array([1,2,3])
print(np.diag(a))
```

```
[[1 0 0]
 [0 2 0]
 [0 0 3]]
```

In [49]:

```
#random.randint()
np.random.randint(1,10)
```

Out[49]:

```
5
```

In [50]:

```
#random.randn()
np.random.randn(1,10)
```

Out[50]:

```
array([[-0.28250199,  0.12522737,  1.24067332,  0.76916943, -0.00682294,
         0.04591771, -2.49103834, -0.44879786, -0.21870954,  0.41260833]])
```

In [51]:

```
#random.random()
np.random.random((3,2))
```

Out[51]:

```
array([[0.56727716, 0.22991477],
       [0.74307796, 0.88894141],
       [0.50561572, 0.29479585]])
```

In [52]:

```
#random.normal()
np.random.normal(2)
```

Out[52]:

```
4.379650168357742
```

In [56]:

```
#index slicing in numpy
a=np.arange(100)
print(a)
print(a[:])
print(a[::2])
print(a[[1,3,4]])
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
[ 0  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46
 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94
 96 98]
[1 3 4]
```

In [79]:

```
a2=np.arange(25).reshape(5,5)
print(a2)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

In [80]:

```
#Task-5:
a2[1:3,3:5]
```

Out[80]:

```
array([[ 8,  9],
       [13, 14]])
```

In [86]:

```
#Task-6:
[a2[1,3],a2[2,4]]
```

Out[86]:

```
[8, 14]
```

In [110]:

```
#Task-7:
[a2[1:5:2,1:5:2]]
```

Out[110]:

```
[array([[ 6,  8],
        [16, 18]])]
```

In [111]:

```
#Task-8:
[a2[1:5:2]]
```

Out[111]:

```
[array([[ 5,  6,  7,  8,  9],
        [15, 16, 17, 18, 19]])]
```

In [ ]: