List of Comprehension

In [1]:

```
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 [2]:

```
list=[i for i in range(1,11)]   #by using list comprehension
list
```

Out[2]:

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

In [3]:

```
def factorial(n):
    if n==0 or n==1:
         return 1
    return n*factorial(n-1)
factorial(5)
```

Out[3]:

```
120
```

In [4]:

```
n=10
list=[factorial(i) for i in range(1,n+1)]
list
```

Out[4]:

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

In [5]:

```
s=["Hai Good Evening"]
li=[]
for i in s:
    for w in i.split():
        li.append(w)
print(li)
```

```
['Hai', 'Good', 'Evening']
```

In [6]:

```
s=["Hai Good Evening"]
list=[w for i in s for w in i.split()]
list
```

Out[6]:

```
['Hai', 'Good', 'Evening']
```

In [8]:

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

Out[8]:

```
15
```

Special Functions in Python

- map()
- filter()
- reduce()
- lambda

In [ ]:

```
#map()

syntax:
    map(functionname,iterator)   #iterations may be list, set, tuple....etc.,
definition:
    Its gives the result after applying given function to each item in the seq
uence
```

In [10]:

```
def square(a):
    return a**a       #(another way: we can use power function-> pow(a,a))
li=[1,2,3,4,5,6]
data=list(map(square,li))
print(data)
```

```
[1, 4, 27, 256, 3125, 46656]
```

In [7]:

```
#map with more parameters
def mulof3(a,b,c):
    return a*b*c
print(list(map(mulof3,[1,2,3],[4,5,6],[7,8,9])))
```

```
[28, 80, 162]
```

In [3]:

```
def char(a):
    a=a.upper()
    b=""
    for i in a:
        b=b+i+" "
    return b
l=list(map(char,[input()]))
print(l[0])
```

```
dileep
D I L E E P
```

In [1]:

```
for i in list(map(str,input().upper())):

    print(i,end=" ").
```

```
sweety
S W E E T Y
```

In [ ]:

```
#filter()

syntax:
    filter(function_name,iter)
```

In [21]:

```
def people(age):
    if age>=18:
        return True
    else:
        return False
voters_list=list(filter(people,[23,18,19,17,15,34,2,3]))
print(voters_list)
```

```
[23, 18, 19, 34]
```

In [22]:

```
#filter Example:

def vowelstrings(var):
    data=["a","e","i","o","u"]
    if var in data:
        return True
    else:
        return False
seq=["n","s","r","i","t","c","o","l","l","e","g","e"]
result=list(filter(vowelstrings,seq))
print(result)
print("filters are: ")
for s in result:
    print(s,end=" ")
```

```
['i', 'o', 'e', 'e']
filters are:
i o e e
```

In [32]:

```
def divisors(val):
    if val%3==0 or val%5==0:
        return True
    else:
        return False
print(list(filter(divisors,[3,6,17,23,78,90,55,44,28,335,227,523])))
```

```
[3, 6, 78, 90, 55, 335]
```

In [41]:
```
#reduce():

from functools import reduce
def mul(x,y):
    return x*y


fact=reduce(mul,range(1,4))
print("factorial of 3: ",fact)
```

```
factorial of 3:  6
```

In [ ]:
```
#lambda with map() and filter and reduce
syntax:
     lambda arg1,arg2:expression
```

In [42]:
```
#map() with a lambda
li=[3,4,5,6,7,8,9]
data=list(map(lambda x:x*2,li))
print(data)
```

```
[6, 8, 10, 12, 14, 16, 18]
```

In [45]:
```
#filter with lambda
li=[23,45,56,78,89,44,12,50]
print(list(filter(lambda x: x%2==0,li)))
```

```
[56, 78, 44, 12, 50]
```

Numpy and Pandas

- It is most commonly used Python Libraries for data sciences
- Numpy name breaks into two parts num+py
- Num is denoted as numrical and py it was represented as python
- it works on array type of data structure
- This package is mainly for scientifc,computing,data analysis

In [4]:
```
import numpy as np
a=np.array([1,2,3])
a
```

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

In [2]:
```
import numpy as np
np.array([[1,2,3],[4,5,6]])
```

Out[2]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [5]:
```
type(a)
```

Out[5]:
```
numpy.ndarray
```

In [7]:
```
a.dtype       #refers the data type of requested argument
```

Out[7]:
```
dtype('int32')
```

In [5]:
```
a=np.array([1,2,3,4.6,6.9,"a"])
print(a)
print(a.dtype)
```

```
['1' '2' '3' '4.6' '6.9' 'a']
<U32
```

In [6]:
```
b=np.array([[1,2,3],[4,5,6]])
print(b)
print(b.shape)            #dimension of b
```

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

In [7]:
```
a[0]
```

Out[7]:
```
'1'
```

In [8]:
```
a
```

Out[8]:
```
array(['1', '2', '3', '4.6', '6.9', 'a'], dtype='<U32')
```

In [9]:
```
a[[0,4,5]]
```

Out[9]:
```
array(['1', '6.9', 'a'], dtype='<U32')
```

In [10]:
```
b[0]
```

Out[10]:
```
array([1, 2, 3])
```

In [53]:
```
b[[1,1]]
```

Out[53]:
```
array([[10, 11, 12],
       [10, 11, 12]])
```

In [14]:
```
a[0:3]
```

Out[14]:
```
array(['1', '2', '3'], dtype='<U32')
```

In [15]:
```
b[0:3]
```

Out[15]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [16]:

```
range(100)
```

Out[16]:

```
range(0, 100)
```

In [19]:

```
np.arange(20,50)
```

Out[19]:

```
array([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])
```

In [20]:

```
np.arange(20,50,2)
```

Out[20]:

```
array([20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48])
```

In [21]:

```
#time of computation

a=range(100)
%timeit [i**5 for i in a]
```

```
28.7 µs ± 150 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [3]:

```
#using numpy for time computation
import numpy as np
b=np.arange(1000)
%timeit b**5
```

```
3.29 µs ± 19.9 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In [6]:

```
c=range(1,1000)
d=[i**2 for i in range (1000)]
%timeit list(map(lambda x,y: x*y,c,d))
```

```
96.3 µs ± 1.61 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [5]:

```
c_array=np.array(c)
d_array=np.array(d)
%timeit c_array*d_array
```

```
1.35 µs ± 3.91 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

In [11]:

```
np.zeros(3)
```

Out[11]:

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

In [13]:

```
np.ones([5,5])
```

Out[13]:

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

```
In [14]: np.eye(4,dtype=int)

Out[14]: array([[1, 0, 0, 0],
                [0, 1, 0, 0],
                [0, 0, 1, 0],
                [0, 0, 0, 1]])
```

```
In [15]: np.diag([1,2,3,4,5])

Out[15]: array([[1, 0, 0, 0, 0],
                [0, 2, 0, 0, 0],
                [0, 0, 3, 0, 0],
                [0, 0, 0, 4, 0],
                [0, 0, 0, 0, 5]])
```

```
In [28]: a=np.array([[1,2,3],[4,5,6]])
         a

Out[28]: array([[1, 2, 3],
                [4, 5, 6]])
```

```
In [27]: a.T      #Transpose

Out[27]: array([[1, 4],
                [2, 5],
                [3, 6]])
```

```
 In [4]: import numpy as np
         a=np.array([[1,2,3],[4,5,6],[7,8,9]])
         np.linalg.inv(a)

 Out[4]: array([[ 3.15251974e+15, -6.30503948e+15,  3.15251974e+15],
                [-6.30503948e+15,  1.26100790e+16, -6.30503948e+15],
                [ 3.15251974e+15, -6.30503948e+15,  3.15251974e+15]])
```

```
 In [5]: np.random.random([2,2])

 Out[5]: array([[0.68149404, 0.26758953],
                [0.55186408, 0.04822421]])
```

```
In [54]: 50*np.random.random([2,2])+3       #Here 3 to 50 range elements are printed ra
         ndomly

Out[54]: array([[48.82171464, 19.14020366],
                [17.26355968,  8.36925326]])
```

```
 In [7]: np.random.randint(1,30)

 Out[7]: 16
```

```
 In [8]: np.random.randint(6)

 Out[8]: 2
```

In [9]:
```
np.random.random(6)
```

Out[9]:
```
array([0.05589128, 0.0645598 , 0.85702093, 0.3962714 , 0.20212485,
       0.60261747])
```

In [10]:
```
a=np.linspace(1,50,30)   #Here 1 to 50 elements are divided into 30 elements
a
```

Out[10]:
```
array([ 1.        ,  2.68965517,  4.37931034,  6.06896552,  7.75862069,
        9.44827586, 11.13793103, 12.82758621, 14.51724138, 16.20689655,
       17.89655172, 19.5862069 , 21.27586207, 22.96551724, 24.65517241,
       26.34482759, 28.03448276, 29.72413793, 31.4137931 , 33.10344828,
       34.79310345, 36.48275862, 38.17241379, 39.86206897, 41.55172414,
       43.24137931, 44.93103448, 46.62068966, 48.31034483, 50.        ])
```

In [55]:
```
a=np.linspace(1,50,3)   #Here 1 to 50 elements are divided into 3 elements
a
```

Out[55]:
```
array([ 1. , 25.5, 50. ])
```

In [57]:
```
#"""Here 2 matrices are created by 3 rows and 3 columns size
#and number of elements should be equal to the size"""

np.arange(18).reshape(2,3,3)
```

Out[57]:
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

       [[ 9, 10, 11],
        [12, 13, 14],
        [15, 16, 17]]])
```

In [12]:
```
np.arange(12).reshape(2,3,-1)
```

Out[12]:
```
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5]],

       [[ 6,  7],
        [ 8,  9],
        [10, 11]]])
```

In [13]:
```
a>2
```

Out[13]:
```
array([False,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True])
```

In [14]:
```
a[(a>2)&(a<5)]
```

Out[14]:
```
array([2.68965517, 4.37931034])
```

In [16]:

```
s1=np.arange(10)
s1
```

Out[16]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [17]:

```
s2=s1         #here swapping of array is taken place
s2
```

Out[17]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [18]:

```
s2[0]=12    # 0 position is replaced by 20 in s2
s2
```

Out[18]:

```
array([12,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

In [19]:

```
s1       #even copying the memory location is remains same for s1
```

Out[19]:

```
array([12,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

In [22]:

```
s3=s1.copy()   #copy method is used to not replace the memory location of s1 a
nd to change the value
s3             #to s3 when any changes happenend to s3 that doesnot effect the
s1 value
```

Out[22]:

```
array([12,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

In [23]:

```
s3[0]=32
s3
```

Out[23]:

```
array([32,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

In [24]:

```
s1
```

Out[24]:

```
array([12,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

In [25]:

```
a=np.array([[1,2,3],[4,5,6]])
b=np.array([[7,8,9],[10,11,12]])
a
```

Out[25]:

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

In [26]:

```
b
```

Out[26]:

```
array([[ 7,  8,  9],
       [10, 11, 12]])
```

In [29]:

```
a+b
```

Out[29]:

```
array([[ 8, 10, 12],
       [14, 16, 18]])
```

In [30]:
```
np.vstack((a,b))          #here matrix is printed vertically
```

Out[30]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [31]:
```
np.hstack((a,b))      #here matrix is printed horizontally
```

Out[31]:
```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

In [32]:
```
np.tan(a)        #trignometric   "tan' value
```

Out[32]:
```
array([[ 1.55740772, -2.18503986, -0.14254654],
       [ 1.15782128, -3.38051501, -0.29100619]])
```

In [33]:
```
np.tan(45)
```

Out[33]:
```
1.6197751905438615
```

In [34]:
```
np.exp(a)      #exponential
```

Out[34]:
```
array([[  2.71828183,   7.3890561 ,  20.08553692],
       [ 54.59815003, 148.4131591 , 403.42879349]])
```

In [35]:
```
np.sqrt(a)  #square root
```

Out[35]:
```
array([[1.        , 1.41421356, 1.73205081],
       [2.        , 2.23606798, 2.44948974]])
```

In [36]:
```
np.std(a)
```

Out[36]:
```
1.707825127659933
```

In [37]:
```
np.median(a)    #median
```

Out[37]:
```
3.5
```

In [38]:
```
np.sum(a+b,axis=0)      #axis=0 reads on coloumn
```

Out[38]:
```
array([22, 26, 30])
```

In [39]:
```
np.sum(a+b,axis=1)      #axis=1 reads on row
```

Out[39]:
```
array([30, 48])
```

```
In [40]: np.dot(a,b)   #it should have to be a square matrix
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-40-3339b236d1c5> in <module>
----> 1 np.dot(a,b)

ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
```

```
In [41]: a%b
```

```
Out[41]: array([[1, 2, 3],
                [4, 5, 6]], dtype=int32)
```

```
In [42]: np.max(a)
```

```
Out[42]: 6
```

```
In [43]: np.min(a)
```

```
Out[43]: 1
```

```
In [45]: np.argmax(a)
```

```
Out[45]: 5
```

```
In [49]: a=([[1,2,3],[4,5,6],[7,8,9]])
         a
```

```
Out[49]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
In [51]: np.linalg.det(a)  #linalg=linear algebra
```

```
Out[51]: -9.51619735392994e-16
```

Pandas

- Pandas is used to creating Dataframes
- It is a open Source library that produces high perfomance and manipulation analysis too
- using pandas we can accomplish live typical steps in data processing
  - data load
  - prepare
  - manipulate
  - model
  - analyze

```
In [58]: import pandas as pd
         pd.__version__
```

```
Out[58]: '0.25.1'
```

Pandas Data Structure

```
- Series
- Data Frames
```

->Series

```
In [60]: pd.Series([1,2,3])
```

```
Out[60]: 0    1
         1    2
         2    3
         dtype: int64
```

```
In [63]: df=pd.Series([1,2,3],index=["a","b","c"])
```

```
In [64]: print(df[0],df[1],df[2])
```

```
         1 2 3
```

```
In [70]: print(df['a'],df['b'],df['c'])
```

```
         1 2 3
```

```
In [67]: df['a':'c']
```

```
Out[67]: a    1
         b    2
         c    3
         dtype: int64
```

```
In [76]: marks={'Maths':100,'Hindi':99,'Telugu':92}
         marks
```

```
Out[76]: {'Maths': 100, 'Hindi': 99, 'Telugu': 92}
```

```
In [77]: pd.Series(marks)
```

```
Out[77]: Maths     100
         Hindi      99
         Telugu     92
         dtype: int64
```

In [5]:

```
dates=pd.date_range('04-7-2019','22-07-2019')
dates
```

Out[5]:

```
DatetimeIndex(['2019-04-07', '2019-04-08', '2019-04-09', '2019-04-10',
               '2019-04-11', '2019-04-12', '2019-04-13', '2019-04-14',
               '2019-04-15', '2019-04-16',
               ...
               '2019-07-13', '2019-07-14', '2019-07-15', '2019-07-16',
               '2019-07-17', '2019-07-18', '2019-07-19', '2019-07-20',
               '2019-07-21', '2019-07-22'],
              dtype='datetime64[ns]', length=107, freq='D')
```

In [6]:

```
x=pd.date_range('04-7-2019',periods=1)
x
```

Out[6]:

```
DatetimeIndex(['2019-04-07'], dtype='datetime64[ns]', freq='D')
```

In [7]:

```
temperature={'04-07-1999':22,'22-01-2001':100}
temperature
```

Out[7]:

```
{'04-07-1999': 22, '22-01-2001': 100}
```

In [8]:

```
pd.Series(temperature)
```

Out[8]:

```
04-07-1999     22
22-01-2001    100
dtype: int64
```

In [12]:

```
import numpy as np
x=pd.Series(np.arange(10,20))
x
```

Out[12]:

```
0    10
1    11
2    12
3    13
4    14
5    15
6    16
7    17
8    18
9    19
dtype: int32
```

In [ ]: