Functional Programming

- Comprehensions in python
  - list Comprehension
  - Tuple Comprehension
  - dictionary Comprehension
  - set Comprehension

- List Comprehension:
- by using list Comprehension we can create one new list from the other iterables.
- Syntax:
  - [output_expression for loop if(condition)]

In [2]:
```
# 1. generate natural numbers from 1 to 20 using list?
#output : [1,2,3,4,5,6,.....20]
l = []
for i in range(1,21):
    l.append(i)
print(l)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

In [3]:
```
[i for i in range(1,21)]
```

Out[3]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

In [5]:
```
# to print even numbers with in range of  1 to 100 using LC

k = [i for i in range(1,101) if i%2==0]
print(k)
```

```
[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, 5
8, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100]
```

In [6]:
```
def odd(n):
    return n%2!=0
s = [i for i in range(1,101) if(odd(i))]
print(s)
```

```
[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, 5
7, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99]
```

In [8]:
```
# to generate prime numbers with in the range of 1 to 100 using LC
prime_list = [x for x in range(0 ,100) for y in range(2,x) if x % x == 0 and x % 1 == 0 and x % y != 0]
print(prime_list)
```

...

In [9]:
```
def prime(n):
    c=0
    for i in range(1,n+1):
        if n%i==0:
            c+=1
    if c==2:
        return n
prime1 = [i for i in range(1,101) if prime(i)]
print(prime1)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

In [10]:
```
n = 100

primes = [i for i in range(2, n + 1) if all(i%j != 0 for j in range(2, int(i ** 0.5) + 1))]

print(primes)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

- Dictionary Comprehension:
  - syntax:
    - {key:value for loop if(condition)}

In [11]:
```
# to generate squares of numbers between 1 to 10 in DC?
# output:{1:1,2:4,3:9,4:12,.....10:100}
d={}
for i in range(1,11):
    d[i]=i**2
print(d)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

In [12]:
```
{i:i**2 for i in range(1,11)}
```

Out[12]:
```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

In [15]:
```
l = ["apple","banana","mango"]
# using above list to print length of given words using DC?
#output:{"apple":5,"banana":6,"mango":5}
dict = {}
List=["apple", "banana", "mango"]
for i in List:
     dict[i] =len(i)
print(dict)
```

```
{'apple': 5, 'banana': 6, 'mango': 5}
```

In [17]:
```
{i:len(i) for i in l}
```

Out[17]:
```
{'apple': 5, 'banana': 6, 'mango': 5}
```

In [21]:
```
l= [10,20,30,10,20,60,30,40,30]
# print count of each item by using DC?
#l.count(element)
#output: {10:2,20:2...}
{i:l.count(i) for i in l}
```

Out[21]:
```
{10: 2, 20: 2, 30: 3, 60: 1, 40: 1}
```

In [ ]:
```
# task1: create a dictionary of characters frequency in a given string by using Dictionary Compreshion?
'''
s = "python programming"
output :
{"p":2,"y":1,"t":1.....}
'''
# task2: DC
#d1 = {"alekhya":22,"aihika":35,"chandana":42}
#only print even values in the item
#output:{"alekhya":22,"chandana":42}

#task3: LC
# input :
'''
"apssdc1234567srkit4567"
output:["a","p","s","s","d","c","s","r","k","i","y"]
'''

# task4 : using LC print multiplication table?
```

In [25]:
```
S = "python programming"
d = {char:S.count(char) for char in S}
print(d)
```

```
{'p': 2, 'y': 1, 't': 1, 'h': 1, 'o': 2, 'n': 2, ' ': 1, 'r': 2, 'g': 2, 'a': 1, 'm': 2, 'i': 1}
```

- Tuple comprehension:
  - syntax:
    - tuple((output_expression for loop condition))

In [26]:
```
# to genereate 1 to 10 numbers using tuple comprehension?
tuple((i for i in range(1,11)))
```

Out[26]:
```
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

- set comprehension:
  - syntax:
    - {output_expression for loop condition}

In [27]:
```
l = [10,20,10,30,50,20]
#{10,20,30,50}

s = {i for i in l}
print(s)
```

```
{10, 20, 50, 30}
```

In [28]:
```
s = "pythonprogramming"
s1 = {i for i in s}
print(s1)
```

```
{'o', 'y', 'p', 't', 'n', 'm', 'a', 'r', 'h', 'i', 'g'}
```

- lambda:
  - by using "lambda" keyword we can develop the function.
  - it is a anynomous function in a single line.
- syntax:
  - z = lambda arguments:expression
  - z(arguments)

In [33]:
```
# to print addition of 2 numbers

a = lambda x,y,z:x+y+z
a(10,20,30)
```

Out[33]: 60

In [37]:
```
# print odd numbers using lambda
b = lambda x:x%2!=0
l = [i for i in range(1,100) if b(i)]
print(l)
```

```
[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, 5
7, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99]
```

- filter:
  - syntax:
    - filter(function_name,sequence)

In [45]:
```
l = [10,-6,23,0,-3,-723,67,-8]
# print only negative values

def negative(n):
    return n<0
print(filter(negative,l))
print(tuple(filter(negative,l)))
print(list(filter(negative,l)))
print(set(filter(negative,l)))
```

```
<filter object at 0x0000024280CA4A20>
(-6, -3, -723, -8)
[-6, -3, -723, -8]
{-8, -6, -723, -3}
```

- map:
  - syntax:
    - map(function_name,sequence)

In [47]:
```
l = [10,-6,23,0,-3,-723,67,-8]
list(map(lambda n:n+10,l))
```

Out[47]: [20, 4, 33, 10, 7, -713, 77, 2]

In [66]:
```
a=list(map(int,"10 20 30".split()))
print(a)
```

```
[10, 20, 30]
```

In [61]:
```
a = "10 20 30"
d = a.split()
print(d)
l=[]
for i in d:
    l.append(int(i))
print(l)
```

```
['10', '20', '30']
[10, 20, 30]
```

- iterator:
  - to fetch the one value at a time
  - syntax:
    - variable_name = iter(iterable)
    - next(variable_name)

In [67]:
```
l = [10,20,30,40]
a = iter(l)
next(a)
```

Out[67]: 10

In [68]:
```
next(a)
```

Out[68]: 20

In [69]:
```
next(a)
```

Out[69]: 30

In [70]:
```
next(a)
```

Out[70]: 40

In [71]:
```
next(a)
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-71-15841f3f11d4> in <module>
----> 1 next(a)

StopIteration:
```

- Generator
- to generate the sequence of values from the function using "yield" keyword

In [74]:
```
def natural(n):
    while n<=5:
        return n

natural(4)
```

Out[74]: 4

In [82]:
```
def natural1(n):
    while n<=5:
        yield n
        n=n+1

list(natural1(1))
```

Out[82]: [1, 2, 3, 4, 5]

In [85]:
```
lc = [n * 18 for n in range(1, 20)]
print (lc)
```

```
[18, 36, 54, 72, 90, 108, 126, 144, 162, 180, 198, 216, 234, 252, 270, 288, 306, 324, 342]
```

In [ ]: