1.1 Write a Python Program to implement your own myreduce() function which works exactly like Python's built-in function reduce()

In [104…
```
# custom implementation of python reduce function

def myreduce(func, values):
    start = values[0]
    for i in values[1:]:
        start = func(start, i)
    return start
```

In [105…
```
# function for multiplying numbers

def multiply(a, b):
    return a * b
```

In [106…
```
# pyhton calling inbuilt reduce function

from functools import reduce
reduce(multiply, [1, 2, 3, 4])
```

Out[106… 24

In [107…
```
# calling custom reduce function

myreduce(multiply, [1, 2, 3, 4])
```

Out[107… 24

1.2 Write a Python program to implement your own myfilter() function which works exactly like Python's built-in function filter()

In [108…
```
# custom implementation of python filter function

def myfilter(func,values):
    for item in values :
        if func(item):
            yield item
```

In [109…
```
# function to determine even numbers

def iseven(x):
    if x % 2 == 0 :
        return True
    else :
        return False
```

In [110…
```
# pyhton calling inbuilt filter function
```

```
f = filter(iseven, [1, 2, 3, 4])

for i in f:
    print(i)
```

```
2
4
```

In [111…
```
# calling custom filter function

mf = myfilter(iseven, [1, 2, 3, 4])

for i in mf:
    print(i)
```

```
2
4
```

2. Implement List comprehensions to produce the following lists.

['A', 'C', 'A', 'D', 'G', 'I', ’L’, ‘ D’]

In [112…
```
lstAlp = ['A','C']
lstAlp.append('A')
```

In [113…
```
lstAlp
```

Out[113… `['A', 'C', 'A']`

In [114…
```
lstAlp.extend('DGILD')
lstAlp
```

Out[114… `['A', 'C', 'A', 'D', 'G', 'I', 'L', 'D']`

['x', 'xx', 'xxx', 'xxxx', 'y', 'yy', 'yyy', 'yyyy', 'z', 'zz', 'zzz', 'zzzz']

In [115…
```
matrix =[['x','y','z'],['xx','yy','zz'],['xxx','yyy','zzz'],['xxxx','yyyy','zzzz']]

first_col = [row[0] for row in matrix]

second_col = [row[1] for row in matrix]

third_col = [row[2] for row in matrix]

lst_patt = []

lst_patt.extend(first_col)
lst_patt.extend(second_col)
lst_patt.extend(third_col)

lst_patt
```

Out[115… ['x', 'xx', 'xxx', 'xxxx', 'y', 'yy', 'yyy', 'yyyy', 'z', 'zz', 'zzz', 'zzzz']

['x', 'y', 'z', 'xx', 'yy', 'zz', 'xxx', 'yyy', 'zzz', 'xxxx', 'yyyy', 'zzzz']

In [116…
```
lst = []

for i in range(4):
    lst.extend(matrix[i])

lst
```

Out[116… ['x', 'y', 'z', 'xx', 'yy', 'zz', 'xxx', 'yyy', 'zzz', 'xxxx', 'yyyy', 'zzzz']

[[2], [3], [4], [3], [4], [5], [4], [5], [6]]

In [117…
```
lstnum = []

for i in range (2,5):
    for j in range(i,i+3):
        lstnum.append([j])

lstnum
```

Out[117… [[2], [3], [4], [3], [4], [5], [4], [5], [6]]

[[2, 3, 4, 5], [3, 4, 5, 6], [4, 5, 6, 7], [5, 6, 7, 8]]

In [118…
```
lst1 = []

for i in range (2,6):
    lst2 = []
    for j in range(i,i+4):
        lst2.append(j)
    lst1.append(lst2)
lst1
```

Out[118… [[2, 3, 4, 5], [3, 4, 5, 6], [4, 5, 6, 7], [5, 6, 7, 8]]

[(1, 1), (2, 1), (3, 1), (1, 2), (2, 2), (3, 2), (1, 3), (2, 3), (3, 3)]

In [119…
```
lstnumpattern = []

for i in range (1,4):
    for j in range(1,4):
        lstnumpattern.append(tuple([j,i]))

lstnumpattern
```

Out[119… [(1, 1), (2, 1), (3, 1), (1, 2), (2, 2), (3, 2), (1, 3), (2, 3), (3, 3)]

In [ ]: