-: Practical Set – 6 :-

1. Write a program to find the prime numbers in a specific range using filter.

```
import sympy
n = 10
l1 = list(range(1,n))
ans = list(filter(lambda x : sympy.isprime(x), l1))
print(ans)
```

OUTPUT:

```
[2, 3, 5, 7]
```

2. Write a python program to make sum of particular range using reduce.

```
import functools
n = 10
l1 = list(range(1, n))
print(l1)
print(functools.reduce(lambda x, y: x+y, l1))
```

OUTPUT:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
45
```

3. Write a python program to find maximum from a list using reduce.

```
import functools
l1 = [1,2,5,7,6,11,65,3,30,24]
print(functools.reduce(lambda x,y : x if x>y else y, l1))
```

OUTPUT:

```
65
```

4. Write a python program to find Armstrong number in a specific range using map.

```
def is_armstrong(number):
    num_digits = len(str(number))
    digit_sum = sum(map(lambda x: int(x)**num_digits, str(number)))
    return digit_sum == number
start = 100
end = 1000
armstrong_numbers = list(filter(is_armstrong, range(start, end+1)))
print(armstrong_numbers)
```

OUTPUT:

```
[153, 370, 371, 407]
```

5. Write a python program to apply two functions (square and cube) simultaneously on a specific range using map.

```
def square(x):
    return x**2
def cube(x):
    return x**3
start = 1
end = 5
squared_and_cubed_values = list(map(lambda x: (square(x), cube(x)), range(start, end+1)))
print(squared_and_cubed_values)
```

OUTPUT:

```
[(1, 1), (4, 8), (9, 27), (16, 64), (25, 125)]
```

6. Write python programs using (i) map/filter and function (ii) map/filter and lambda (iii) list comprehension.

- Create a list to store the cube of all the elements in a given list.

```
def cube(num):
    return num**3

given_list = [1, 2, 3, 4, 5]
cubed_list = list(map(cube, given_list))
print("Using map/filter with function : ", cubed_list)
cubed_list = list(map(lambda x: x**3, given_list))
print("Using map/filter with lambda : ", cubed_list)
cubed_list = [num**3 for num in given_list]
print("Using List Comprehension : ", cubed_list)
```

OUTPUT :

```
Using map/filter with function :  [1, 8, 27, 64, 125]
Using map/filter with lambda :  [1, 8, 27, 64, 125]
Using List Comprehension :  [1, 8, 27, 64, 125]
```

- Create a list of equivalent Celsius degree from Fahrenheit.

```
def f_to_c(deg_f):
    return (deg_f - 32) * 5/9

deg_f_list = [32, 68, 86, 104, 122]
deg_c_list = list(map(f_to_c, deg_f_list))
print("Using map/filter with function : ", deg_c_list)
deg_c_list = list(map(lambda f: (f - 32) * 5/9, deg_f_list))
print("Using map/filter with lambda : ", deg_c_list)
deg_c_list = [(deg_f - 32) * 5/9 for deg_f in deg_f_list]
print("Using List Comprehension : ", deg_c_list)
```

OUTPUT :

```
Using map/filter with function :  [0.0, 20.0, 30.0, 40.0, 50.0]
Using map/filter with lambda :  [0.0, 20.0, 30.0, 40.0, 50.0]
Using List Comprehension :  [0.0, 20.0, 30.0, 40.0, 50.0]
```

- Create a list that stores only positive numbers from given list.

```
def is_positive(num):
    return num > 0

given_list = [1, -2, 3, -4, 5]
positive_list = list(filter(is_positive, given_list))
print("Using map/filter with function : ", positive_list)

positive_list = list(filter(lambda x: x > 0, given_list))
print("Using map/filter with lambda : ", positive_list)

positive_list = [num for num in given_list if num > 0]
print("Using List Comprehension : ", positive_list)
```

OUTPUT :

```
Using map/filter with function :  [1, 3, 5]
Using map/filter with lambda :  [1, 3, 5]
Using List Comprehension :  [1, 3, 5]
```

- Create a list that stores only alphabets from given list

```
def is_alpha(char):
    return char.isalpha()

given_list = ['a', '1', 'B', '2', 'c', '3']
alpha_list = list(filter(is_alpha, given_list))
print("Using map/filter with function : ", alpha_list)

alpha_list = list(filter(lambda x: x.isalpha(), given_list))
print("Using map/filter with lambda : ", alpha_list)

alpha_list = [char for char in given_list if char.isalpha()]
print("Using List Comprehension : ", alpha_list)
```

OUTPUT :

```
Using map/filter with function :  ['a', 'B', 'c']
Using map/filter with lambda :  ['a', 'B', 'c']
Using List Comprehension :  ['a', 'B', 'c']
```