Plan of Action

- Lambda functions
- Map functions
- reduce
- filter
- recursion

Lambda

A lambda function is a small anonymous functions defined with the lambda keyword.It can have any number of arguments but only one expression.It's often used for a short,simple function that are needed temporarily.

In [38]:
```
# aam zindagi
def sum(x,y):
    return(x+y)
```

In [39]:
```
# mentos zindagi
# lambda function
sum= lambda x,y : x+y
```

In [40]:
```
sum(109,20)
```

Out[40]:
```
129
```

In [41]:
```
# single arguements
# cube
cube = lambda x:x**3
```

In [42]:
```
cube(3)
```

Out[42]:
```
27
```

In [43]:
```
# 3-4
mean_of_3=lambda x,y,z: (x+y+z)/3
```

In [44]:
```
mean_of_3(10,12,236)
```

Out[44]:
```
86.0
```

In [45]:
```
# lambda function--? no arguements
greet = lambda :  'Hello world'
greet()
```

Out[45]: 'Hello world'

In [46]:
```
# conditioned based
# even odd
tell = lambda x: 'even' if x%2==0 else 'odd'
```

In [47]:
```
tell(2)
```

Out[47]: 'even'

In [48]:
```
tell(1)
```

Out[48]: 'odd'

In [49]:
```
# positive negative
tell2 = lambda x: 'positive' if x>=0 else 'negative'
```

In [50]:
```
tell2(12)
```

Out[50]: 'positive'

In [51]:
```
tell2(-8)
```

Out[51]: 'negative'

Map Functions

The map function applies a given function to all items in an input list (or any other iterable) and return a map objec(which is an iterator).You can convert object into list or other data structure is needed.

In [52]:
```
# aam zindagi
l1=[1,2,3,4,5,56,783,23,23]
square= lambda x:x**2
# square(l1)
n1=[]
for i in l1:
    # print(square(i))
    n1.append(square(i))

n1
```

Out[52]: [1, 4, 9, 16, 25, 3136, 613089, 529, 529]

In [53]:
```
list(map(square,l1)) #mentos zindagi
```

Out[53]: [1, 4, 9, 16, 25, 3136, 613089, 529, 529]

In [54]:
```
l1=[2,5,7,10] #--> cube
list(map(lambda x:x**3,l1))
```

Out[54]:
```
[8, 125, 343, 1000]
```

In [55]:
```
# even odd
list(map(lambda x: 'even' if x%2==0 else 'odd',l1))
```

Out[55]:
```
['even', 'odd', 'odd', 'even']
```

In [56]:
```
# anonymous functions

sum(10,20)
```

Out[56]:
```
30
```

In [57]:
```
(lambda x,y:x+y)(10,20)
```

Out[57]:
```
30
```

Reduce

The reduce function is part of the functools module. It applies a rolling computation a sequential pairs of values in an iterable. It performs a repetative operation over the pairs of the iterable.

syntax

reduce(function, iterable)

In [59]:
```
# reduce
from functools import reduce
```

In [60]:
```
# aam zindagi
l1=[1,2,3,4,5,6]
sum=0
for i in l1:
    sum+=i
sum
```

Out[60]:
```
21
```

In [61]:
```
# mentos zindagi
reduce(lambda x,y:x+y,l1)
```

Out[61]:
```
21
```

Filter

The filter constructs an iterator from elements of an iterable if an iterable for which a funtion returns true. it Filters out the elements that do not satisfy the condition provided by the function.

- filter(function, iterable)

In [62]:
```
# filter
l1 = list(range(0,50,3))
l1
```

Out[62]:
```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48]
```

In [63]:
```
list(filter(lambda x: True if x>25 else False,l1))
```

Out[63]:
```
[27, 30, 33, 36, 39, 42, 45, 48]
```

Recursion

Recursion in Python is a programming technique where a function calls itself directly or indirectly to solve a problem. This approach is particularly useful for problems that can be naturally broken down into smaller, self-similar subproblem

In [64]:
```
# factorial
# 5! = 5*4*3*2*1
def factorial(n):
    if n==0 or n==1:
        return 1
    else:
        return n*factorial(n-1)
factorial(5)
```

Out[64]:
```
120
```

In [65]:
```
# 1! = 1
# 0! = 1
```

In [66]:
```
n = int(input('enter a number'))
prod = 1
for i in range(1,n+1):
    prod*=i
prod
```

Out[66]:
```
120
```

In [ ]: