Inbuilt functions of Py

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

SYNTAX :

```
map(FUNCTION_NAME, ITERATOR)

function name  can be any function :given by function name without parenthesis
iterator is a sequence traveller   :given as a list/tuple/set
```

In [1]:
```
a = list(map(int,input().split()))
```

```
1 2 3 4 5 6
```

In [2]:
```
a
```

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

In [4]:
```
# mapping with single parameter
def sq(k):
    return (k*k)
li = list(map(sq,a))
li
```

Out[4]:
```
[1, 4, 9, 16, 25, 36]
```

In [5]:
```
l1 = map(lambda x:x*x,a)  #using lambda
l1 = list(l1)
l1
```

Out[5]:
```
[1, 4, 9, 16, 25, 36]
```

In [8]:
```
# mapping with multiple parameter
def mul(a,b,c):
    return a*b*c
print(list(map(mul,[1,2,3],[4,5,6],[7,8,9])))
```

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

In [9]:
```
l2 = map(lambda x,y,z:x*y*z,[1,2,3],[4,5,6],[7,8,9])   #using lambda
l2=list(l2)
l2
```

Out[9]:
```
[28, 80, 162]
```

In [26]:
```
def ch(a):
    a=a.upper()
    b=""
    for i in a:
        b=b+i+" "
    return b
l3 = list(map(ch,[input()]))
print(l3[0])
```

```
dngfpkj
D N G F P K J
```

In [28]:
```
j =input().upper()
for i in tuple(map(str,j)):
    print(i," ",end="")
```

```
kugfw
K   U   G   F   W
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

FILTER FUNCTION FOR EXTRACTING

SYNTAX :

```
filter( Extractor_FUNCTION_NAME , ITERATOR )

Extractor_FUNCTION_NAME is a function that will return a boolean value for each
element that goes in it.
ITERATOR is any sequence that is to be filtered out.
filter() is a function that saperates the extracted value {True} from the resid
ual value {False}.
```

In [14]:
```
def vote(age):
    if age<18:
        return False
    else:
        return True
voter_age = list(filter(vote,[12,34,45,56,12,13,11]))
voter_age
```

Out[14]:
```
[34, 45, 56]
```

In [ ]:

In [15]:
```
#vovel filter by function

def vote(a):
    if a in ["a","e","i","o","u"]:
        return True
    else:
        return False
check = list(filter(vote,"qwederwefivoeurvbnq"))
check
```

Out[15]:
```
['e', 'e', 'e', 'i', 'o', 'e', 'u']
```

In [28]:
```
#vovel filter by lambda



check = list(filter(lambda x : (x in ["a","e","i","o","u"]),input()))
print(check)
```
```
ygtjntebdvaeyt4eruwsati
['e', 'a', 'e', 'e', 'u', 'a', 'i']
```

In [ ]:
```
#task
check = list(filter(lambda x : (x in ["a","e","i","o","u"]),input()))
print(check)
```

In [34]:
```
list(filter(lambda x:x%2==0,range(1,10)))
```
Out[34]:
```
[2, 4, 6, 8]
```

In [42]:
```
list(filter(lambda x:x%2==0,map(int,input().split())))
```
```
1 2 3 4 5 6 7
```
Out[42]:
```
[2, 4, 6]
```

In [44]:
```
list(filter(lambda x:x%2==0,range(1,int(input())+1)))
```
```
7
```
Out[44]:
```
[2, 4, 6]
```

In [49]:
```
exit()
```

REDUCE FUNCTION FOR exponential Incermentation

SYNTAX :

```
reduce( incrementor_FUNCTION_NAME , ITERATOR )

incrementor_FUNCTION_NAME is a function that will return a single value for all
the elements that goes in it.
ITERATOR is any sequence that is to be filtered out.
reduce() is a function that calculates the previous value to the new value
```

In [1]:
```
# use of reduce function
from functools import reduce

def mul(x,y):
    return x*y
fact= reduce(mul,range(1,6))
fact# 1 2 3 4 5
```
Out[1]:
```
120
```

In [2]:
```
fact = reduce(lambda x,y:x+y , range(1,6))
fact
```
Out[2]:
```
15
```

In [3]:
```
import operator as op
add = reduce(op.add,range(0,6))
add
```

Out[3]:
```
15
```

In [7]:
```
mul = reduce(op.mul,range(1,6))
mul
```

Out[7]:
```
120
```

LAMBDA FOR ANNOMOUS FUNCTION

SYNTAX :

```
FUN = lambda ARG_1,ARG_2 : EXPRESSION_OF_THE_TWO_ARGUMENTS

    lambda returns a val ..so it can also be used directly for other special fu
nctions
    it doesnot need to give return as a keyword expression is enough
```

In [10]:
```
fact = reduce(lambda x,y:x+y , range(1,6))
fact
```

Out[10]:
```
15
```

In [11]:
```
list(filter(lambda x:x%2==0,range(1,int(input())+1)))
```

```
8
```

Out[11]:
```
[2, 4, 6, 8]
```

In [12]:
```
list(filter(lambda x:x%2==0,map(int,input().split())))
```

```
1 2 3 4 2 1 3 4 5 6 7 80
```

Out[12]:
```
[2, 4, 2, 4, 6, 80]
```

In [42]:
```
list(filter(lambda x:x%2==0,map(int,input().split())))
```

```
1 2 3 4 5 6 7
```

Out[42]:
```
[2, 4, 6]
```

In [13]:
```
l2 = map(lambda x,y,z:x*y*z,[1,2,3],[4,5,6],[7,8,9])   #using Lambda
l2=list(l2)
l2
```

Out[13]:
```
[28, 80, 162]
```

In [ ]:
```

```