Functional programming

- functional programming is a programming paradigm in which we try to bind everything in pre mathematical functional style
- python supports functional programming

lambda:

- Anonymous function
- syntax: lambda arg1,arg2:expression

In [1]:

```
def add(a,b):
    return a+b
add(78,23)
```

Out[1]:

```
101
```

In [3]:

```
ad=lambda a,b:a+b
ad(45,12)
```

Out[3]:

```
57
```

In [4]:

```
type(ad)
```

Out[4]:

```
function
```

In [5]:

```
# write a lambda function to perform square of a number

sq=lambda n:n**2
sq(4)
```

Out[5]:

```
16
```

In [6]:

```
pow=lambda n1,n2:n1**n2
pow(2,6)
```

Out[6]:

```
64
```

In [7]:

```
# cube of a number

cube=lambda a:a**3
cube(5)
```

Out[7]:

```
125
```

In [8]:

```
# if else in lambda

def numcheck(num):
    if num<20 and num>10:
        return True
    else:
        return False
numcheck(18)
```

Out[8]:

```
True
```

In [9]:

```
numcheck(60)
```

Out[9]:

```
False
```

In [12]:

```
nc=lambda a:True if (a<20 and a>10) else False
print(nc(19))
print(nc(80))
```

```
True
False
```

In [13]:

```
# without using if else

nn=lambda a:(a>10 and a<20)
nn(16)
```

Out[13]:

```
True
```

In [14]:

```
nn(90)
```

Out[14]:

```
False
```

Map()

- map() function takes function and iterable as a parameters and applies functionality to each item in the iterable
- syntax:
- map(function,iterable)

In [17]:

```
for i in range(1,11):
    print(i*i,end=" ")
```

```
1 4 9 16 25 36 49 64 81 100
```

In [22]:

```
def sq(x):
    return x*x

f=map(sq,[10,20,30,40,50])
print(f)
print(list(f))
```

```
<map object at 0x000001A18B0EA0A0>
[100, 400, 900, 1600, 2500]
```

In [26]:

```
# cubes

def cube(x):
    return x*x*x

list(map(cube,[2,4,6,8,10]))

```

Out[26]:

```
[8, 64, 216, 512, 1000]
```

In [25]:

```
list(map(cube,range(1,11)))
```

Out[25]:

```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

In [27]:

```
list(map(lambda x:x*x*x,[1,2,3,4,5]))
```

Out[27]:

```
[1, 8, 27, 64, 125]
```

filter()

- filter takes a function and a sequence and returns an iterable
- filters the given iterable with the help of a function that tests each element in the iterable true or false
- syntax: filter(function,iterable)

In [28]:

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

is_even(90)
```

Out[28]:

```
True
```

In [29]:

```
is_even(99)
```

Out[29]:

```
False
```

In [31]:

```
f=filter(is_even,range(1,51))
list(f)
```

...

In [33]:

```
list(filter(is_even,[1,6,9,8,56,90]))
```

Out[33]:

```
[6, 8, 56, 90]
```

In [34]:

```
lis=['b','u','y','a','i','z','e','r','o']

def is_vowel(l):
    v=['a','e','i','o','u']
    if l in v:
        return True
    else:
        return False

# filter(func,lis)

list(filter(is_vowel,['b','u','y','a','i','z','e','r','o']))
```

Out[34]:

```
['u', 'a', 'i', 'e', 'o']
```

reduce()

- reduce(func,seq) is used to apply a particular function passed in its arguments to all of the list elements in the sequence
- functools module

In [35]:

```
import functools
```

In [37]:

```
l=[1,2,3,4,5]
functools.reduce(lambda a,b:a+b   ,  l)   # 1+2+3+4+5
```

Out[37]:

```
15
```

In [38]:

```
# multiplication of numbers in the range of 1 to 10

functools.reduce(lambda a,b:a*b, range(1,11))
```

Out[38]:

```
3628800
```

In [ ]:

```

```