Short notes

```
T = lambda a,x,b : a*x+b

T(2,4,5)   #prints 13

def conversion(a,c):
    return (lambda x: a*x+c)
mtr_km = conv(0.001, 0) ; mtr_km(2000)
```

Map, Filter & Reduce

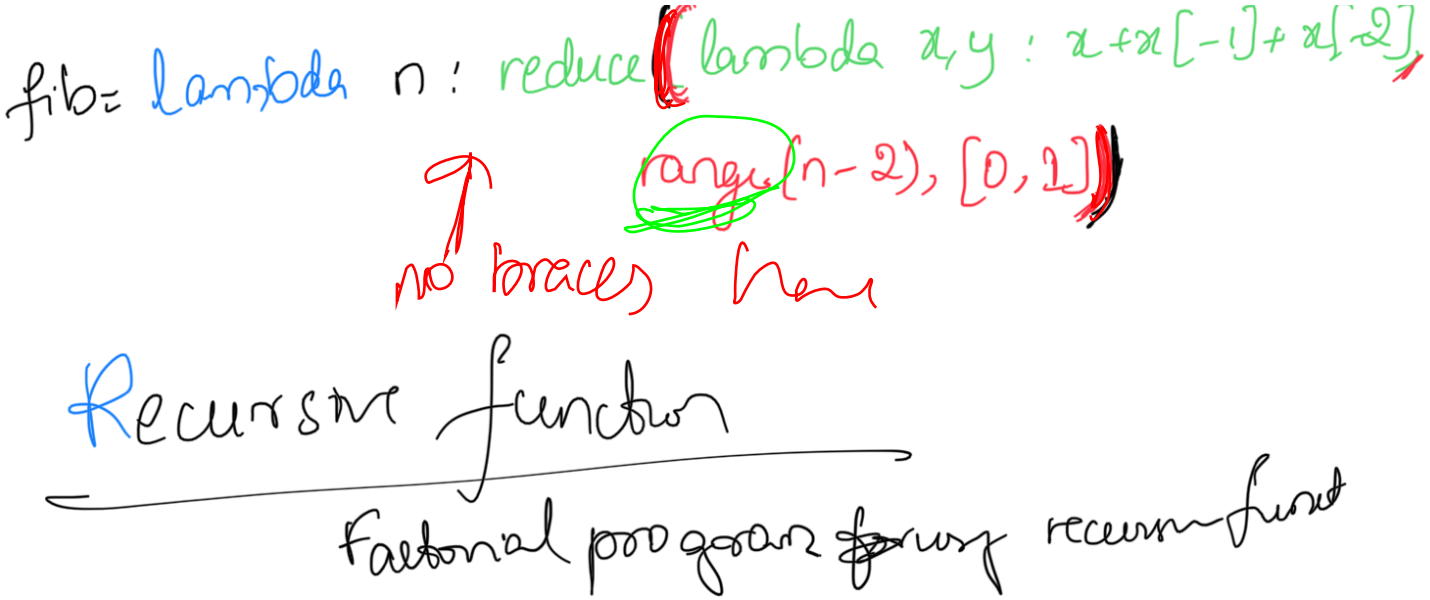
```
l = list(map(lambda x: x**2, l1))

l = list(filter(lambda x: x%2==0, l1))

from functools import reduce

x = reduce(lambda x,y: x+y, l1)
```

Fibonacci series




```
def rec(x):
  if(x > 1):
    res = x * rec(x - 1)
  else:
    res = 1
  return res

print(rec(5))
```

```
120
```

File Handling

import os
os.getcwd()
os.listdir()
#can specify folder inside
#like r'C:\Users\Desktop'

os.path.isfile("FileName.txt')

filePath = r'C:\Desktop\Fname.txt'

File Creation

```
f1 = open('fname.txt', 'x')
f1.close()
```

(no indent), no colon

Easy way to avoid open close

With method

```
with open(r'f_add\fname.txt', 'w') as f2:
    f2.write("zip test")
```

x - mode if file exists, error

indent there, colon

```
try:
except:
```

For file name creation for current time
from datetime import datetime

Timefilename= x.strftime('%d-%m-%**Y**-%H-%M-%S.txt')

with open(**Timefilename**, 'w') as fp
#observe it's a variable
If in a specific folder

file_name = r"C:\Users\LENOVO\OneDrive\Desktop****" + x.strftime('%d-%m-%Y-%H-%M-%S.txt'
#observe double \\

For reading

OOPS : Objects and Classes
note , self. Is needed inside def __init__(self, self.Var)
in just argument passing methods , , directly use , Var instead of self

class Customer :

```
    def __init__(self,name,age) :
            self.custname = name
            self.custage = age
            self.custbalance = 0

            #noteNoBalanceArguementPassedasImNotGivingInput



    def display(self):
            print(self.custname)

    def deposit(self,amount) :
            self.custbalance = self.custbalance + amount
```

```
NirSir = Customer("Niranjan","45")

print(NirSir.custname)
NirSir.display()
NirSir.deposit(500)

#Inheitance Example
class IBcustomer(Customer) :
        def __init__(self,IPF,name,age) :
                Customer.__init__(self,name,age)
                self.IPF = IPF
NirSir = IBcustomer('4Cr',"Niranjan","45")

#Private Members Demonstation
```

class Base:	
	def __init__(self):
	self.a = 'Prime Intuit'
	self.__c = 'PrimeIntuit_Private'
	def display(self):
	print(self.a)
	print(self.__c)

class Derived(Base):	
	def __init__(self):
	Base.__init__(self)
	print("Calling private member of base class: ")
	print(self.a)
	#print(self.__c) #this if not commented will throw erroe for

obj2 derived

obj1 = Base()
obj1.display()
obj2 = Derived() #in this c throws error , like child accesing parent locker
#AttributeError: 'Derived' object has no attribute '_Derived__c'
#calling display from base class , this works ,
obj2.display() #in display can be print

If don't want to pass input , don't declare

def __init__(self):	
	self.balance =

#if want to take input , no need to declare inside init

def opening_account(self):	
	name = input("Enter Acoount holder name: "

Theory to Focus On

Oops

Polymorphism

Types of inheritance

Oops concepts , objects classes to encapsulation

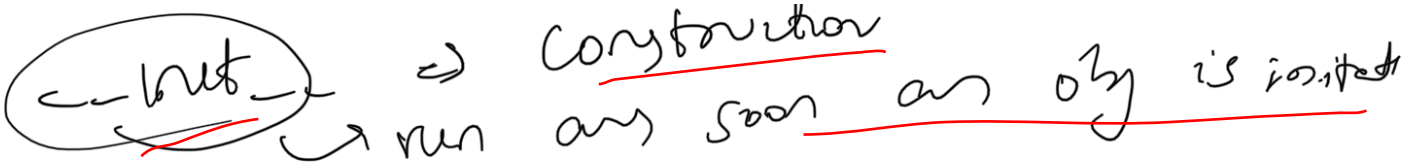
OOPs - programming paradigm that uses objects & classes in programming

- to implement real world entities like inheritance, polymorphism, encapsulation, etc in pr

- Class - architecture / blue print of object
  - description of attributes & methods of class

Object - entity that has state & behaviour associated with it
  - it may be any real world object like mouse, keyboard, chair, table, pen, etc

self - represents instance of class




```
class Dog :
    attr1 = "mammal"
    def __init__(self,name) :
        self.name = name
Rodger = Dog("Rodger")
print("Rodger is a {}".format(Rodger.__class__.attr1))
print("My name is {}".format(Rodger.name))
```

O/p

Rodger is a mammal

My name is Rodger