## **PRocedure Oriented Vs OOP**

C, COBOL

Presentation Logic(View), Business Logic(Controller), Database Logic (Modal)

Logics are Tightly Coupled

**OOProgramig**

Logics are written in **COMPONENT based**

## 

## **Advantage: CODE Reusability**

Java, PYthon, C#, Ada, Apex, PHP, ABAP

## **Object Oriented Concepts**

## **What is class and object**

## Pen redballon system blueballon bottle Yellowballon Paper

## 

## Identify common objects

## 

## redballon blueballon yellowballon

## **Common Attributes (Property)**

## 

## color red blue yellow

## size 20 30 20

## shape oval circle oval

## 

## 

## Common functions (Methods) or Task : fill air:

## function performs on object effects attributes

## 

## 

## **CLASS : Collection of COMMON Attributes and COMMON Functions or Methods**

Logical structure of an OBJECT

Physically (can see): Object

Logically (can’t see only representation) : Class

## 

## class ballon :

## 

## // data members (or) variables

## integer size;

## string shape;

## string color;

## 

## // member functions or methods

## 

## fillair()

## 

## 

## Example::

class emp:

eid = 123

ename = 'anand'

es = 8000

def f1():

print(' I am in f1 ')

print( ' eid = ', eid , 'ename = ',ename)

emp e1

e1.f1()

emp e2

e2.f1()

## Definition:

## **Class is an LOgical structure, how object looks**

**Is collection of data members (Properties) and Methods (Functions)**

**Class is Blue print of an object Ex: building**

## **To write a class :** identify common objects, Common Attributes and Common Methods

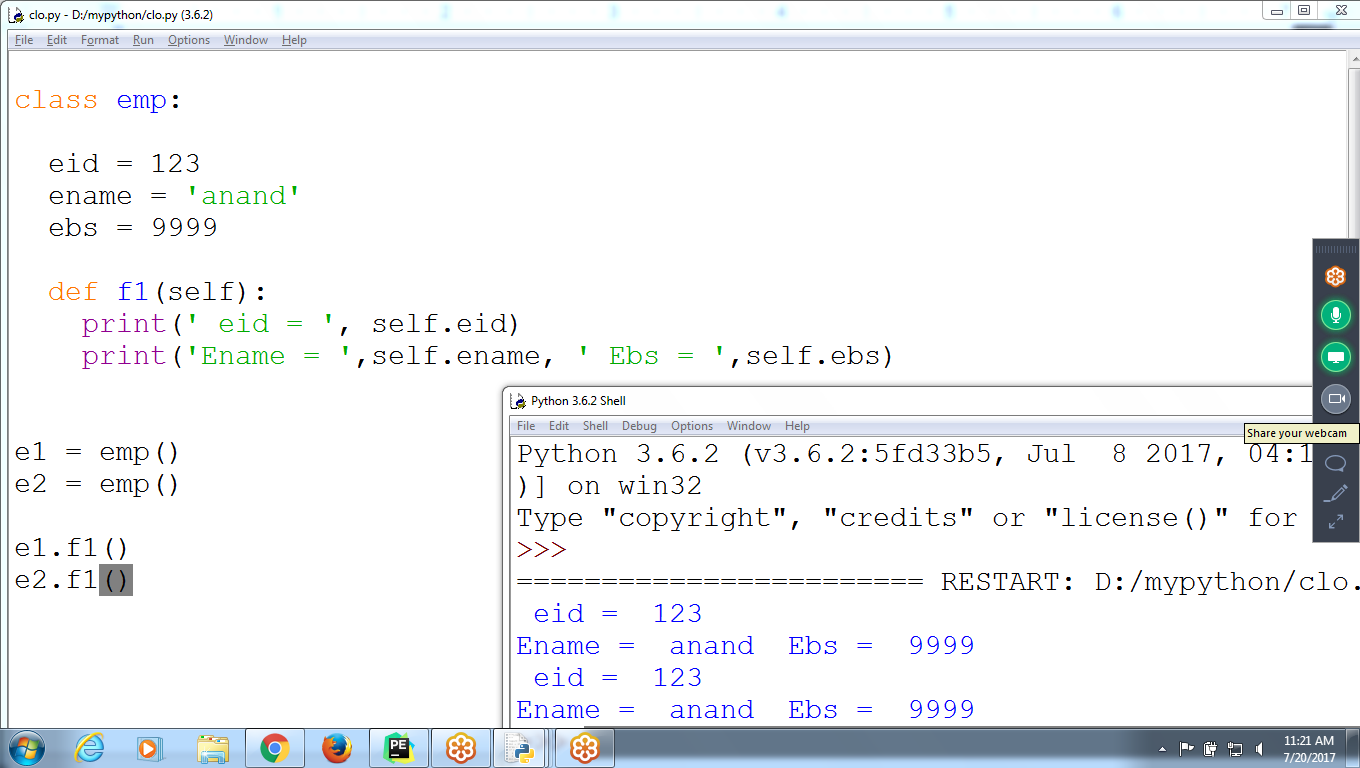
## 

## **Class and Methods will not occupy memory i.e it is an structure**

**Methods are used to operate on data (variables)**

## 

**Program 1: Create a Class**



class emp:

eid = 123

ename = 'anand'

ebs = 9999

def f1(self):

print(' eid = ', self.eid)

print('Ename = ',self.ename, ' Ebs = ',self.ebs)

e1 = emp()

e2 = emp()

e1.f1()

e2.f1()

## **OOP features**

## 

Encapsulation

Polymorphism

Abstraction

Inheritance

## **1 Encapsulation:** Binding data members and methods together is called encapsulation

## i.e **Only methods can process data**

## 

## **Advantage: Security**

## ex: eid, ename can process by f1() method only

## 

class emp:

eid = 123

ename = 'anand'

def f1():

print(' I am in f1 ')

print( ' eid = ', eid , 'ename = ',ename)

emp e1

e1.f1()

emp e2

e2.f1()

## **Polymorphism:**

## Same method name, having different functionality

## 

## **Advantage:** easy to process **with single method name,**

## **no need to remember all method names**

## as per parameter list, return corresponding method calls

## 

## ex: m1(), m1(string), m1(int,string)

## 

Calling : m1(), m1(‘raju’), m1(12, ‘kiran’)

**If NO polymorphism, need to define with different method names**

## 

## # Polymorphism :

## same method name, having multiple definitions

## 

## 

## f1(i,i)

## f2(i,s)

## f3(i,f,st0

## f4(st,i,f)

## 

## 

## **f1(i,i)**

## **f1(i,s)**

## **f1(i,s,f)**

## **f1(i,f,s)**

## 

## 

## f1(2,3)

## f1(34, 'raj')

## 

## 

## 

## **Abstraction:**

## **HIDING How Method is implemented**

## 

## ex:sqrt(4) return 2

## 

## Ex: e1.f1(), e2.f1()

## 

## **Advantages: Reduces complexity**

## Note : Hiding data means :: **Encapsulation (Security)**

Hiding How method is Implemented :: **Abstraction ( Reduce Complexity)**

class emp:

eid = 123

ename = 'anand'

**def f1():**

print(' I am in f1 ')

**print( ' eid = ', eid , 'ename = ',ename)**

emp e1

**e1.f1()**

emp e2

**e2.f1()**

## **Inheritance:**

## 

## Extending or Modifying existing class features / Functionality

## 

## Advantage : **REUsability**

## 

## 

## 