Data Science & AI
&
NIC - Param

Python-For Data Science

OOPs

Lecture No.- 03

By- Pankaj Sharma Sir

Recap of Previous Lecture

Topic Object-Oriented Programming Part -02

Topics to be Covered

Topic Object-Oriented Programming Part -03

Topic : Object-Oriented Programming

10 + 20

PL

- complex number
- vectors
- Fraction $\left(\frac{p}{q}\right)$

2 x 3

2.3 x 4.6

OOPS

- user defined data type

```
Class Complex :

    def __init__(self, real, imag) :
        self.real = real
        self.imag = imag
```

```
c1 = complex(1,2)   # 1+2j
c2 = complex(3,4)   # 3+4j
```

c1 x c2

OOPS

10 + 20

→ user defined data type

PL

→ complex number

→ vectors

→ Fraction $\left(\frac{p}{q}\right)$

Class Complex :

def __init__(self, real, imag) :

self.real = real

self.imag = imag

C1 × C2

C1.multiply(C2)

def multiply(self, compl) :

2 × 3

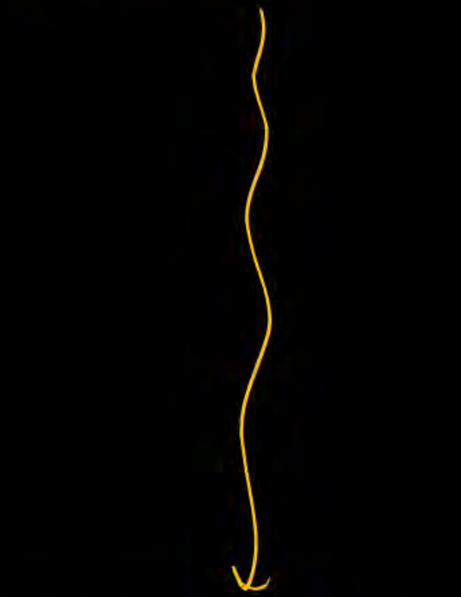
2.3 × 4.6

class matrix :

def determinant( )

SDE1

Class ——————



a = Account()

a.minbalance = 0

1000

SDE1

Class Account :

minbalance ⇒ private

complex (a, b) ⇒

fraction (a, b) ⇒

def normalize(self):

GCD (num, Deno)

$$\frac{9}{3} = \frac{3}{1}$$

14/6 ⇒

15 , 33

14 | 42

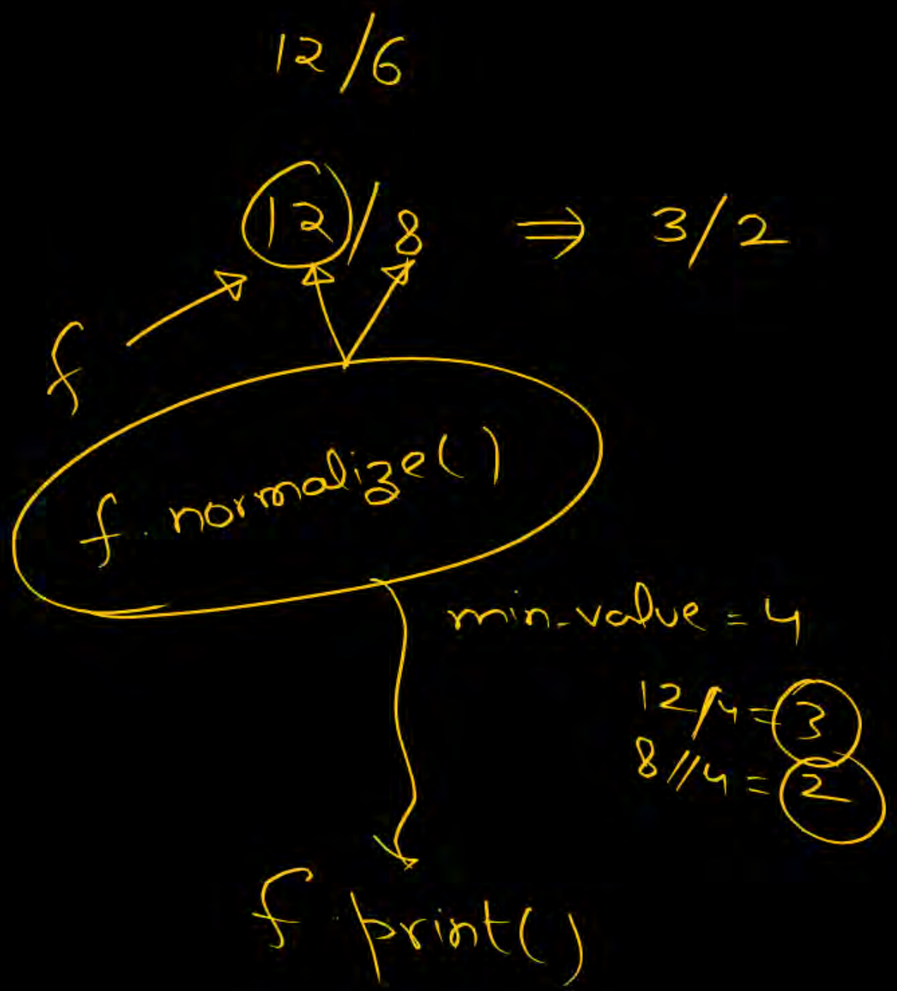
$GCD(a,b) \leq \min(a,b)$

for i — 1, 14 :

14 ⟶ 1

for i — 14, 1 :

(Largest first)


12/6
12/8 ⇒ 3/2
f
f.normalize()
min_value = 4
12//4 = 3
8//4 = 2
f.print()

$\frac{4}{5} - \frac{1}{2}$

f1 f2

f1.subtract(f2)

def subtract(self, doosra_f):

$\frac{a}{b} - \frac{c}{d}$

$\frac{ad - cb}{bd}$

$\frac{5}{6} - \frac{1}{6}$

$\frac{4}{6} \Rightarrow \frac{2}{3}$

up_num = self.num × doosra_f.deno
− doosra_f.num × self.deno

up_deno = self.deno × doosra_f.deno

self.num = up_num
self.deno = up_deno
self.normalize()

Inheritance → DA

Parent — Evil

attributes

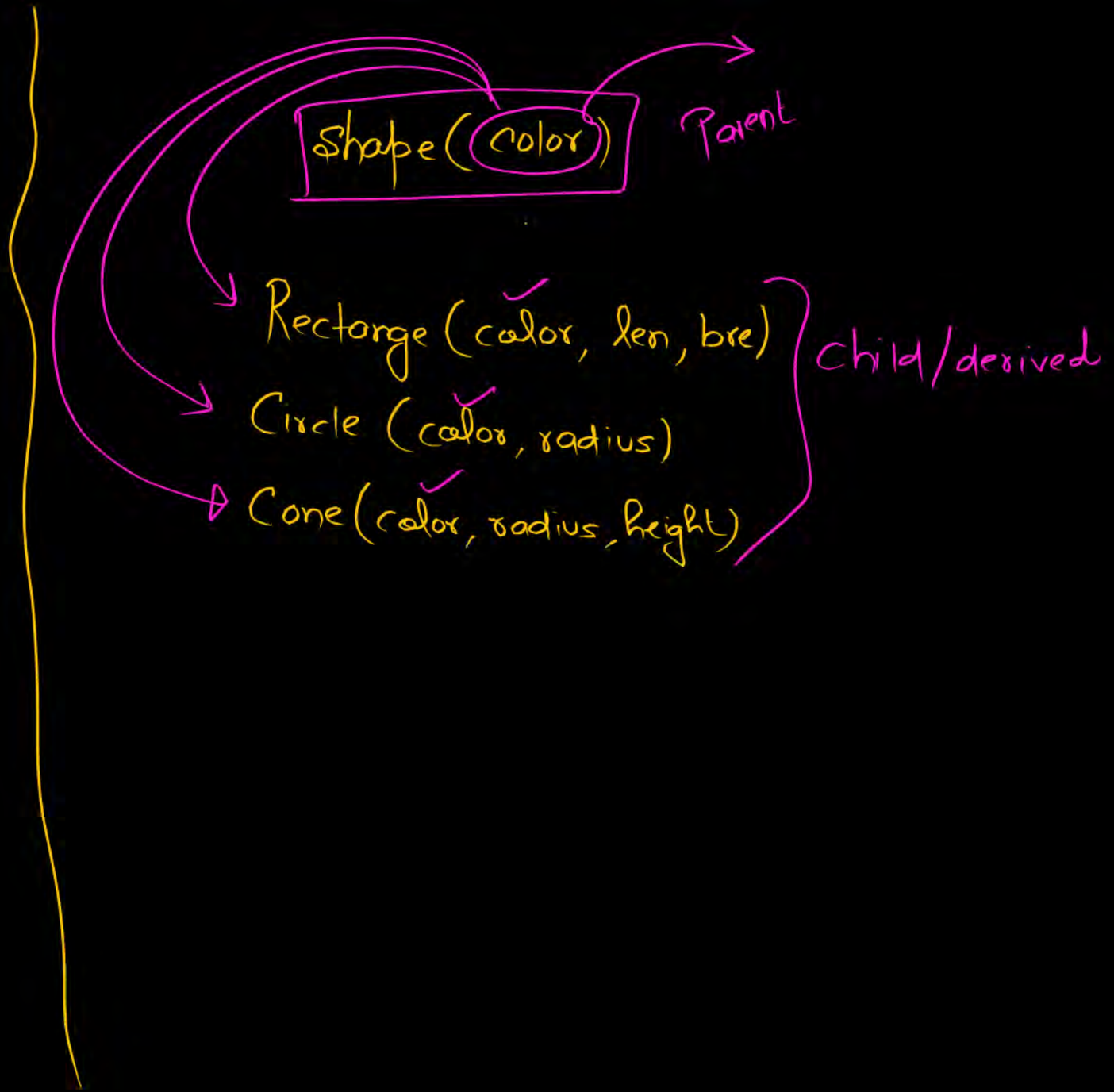
Child

Vehicle (color, speed)

Car (Color, speed, Number_tyres)

Vehicle

inheritance



Linked list

OOPS

Security

Abstraction / Encapsulation

data + methods

Pankaj

variables

private

method

public

a.

Day 23

In [1]:
```
class demo:
    def __init__(self):
        self.x=12
        self.y=90
d=demo()
print(d.x)
print(d.y)
```

```
12
90
```

In [2]:
```
class demo:
    def __init__(self):
        self.x=12
        self.y=90
d=demo()
d.__dict__
```

Out[2]:
```
{'x': 12, 'y': 90}
```

In [3]:
```
class demo:
    def __init__(self):
        self.x=12
        self.y=90
    def fun(self):
        self.z=100
d1=demo()
d2=demo()
d1.__dict__
```

Out[3]:
```
{'x': 12, 'y': 90}
```

In [4]:
```
d2.__dict__
```

Out[4]:
```
{'x': 12, 'y': 90}
```

In [5]:
```
d1.fun()
```

In [6]:
```
d1.__dict__
```

Out[6]:
```
{'x': 12, 'y': 90, 'z': 100}
```

In [7]:
```
d2.__dict__
```

Out[7]:
```
{'x': 12, 'y': 90}
```

In [11]:
```
class emp:
    def __init__(self):
        self.__name='pankaj'    #private
        self.age=40
```

```
    def marriage(self):
        self.wife='Abhilasha'
```

In [13]:
```
e1=emp()
print(e1.age)
```

```
40
```

In [14]:
```
class demo:
    def __init__(self):
        self.x=12
        self.y=90
    def fun(self):
        self.z=100
```

In [21]:
```
class Complex:
    def __init__(self,real,imag) :
        self.real=real
        self.imag=imag
    def print(self):
        print(self.real, "+",self.imag,"i")

c=Complex(2,3) #real,imag
```

In [22]:
```
c.real
```

Out[22]:
```
2
```

In [23]:
```
c.imag
```

Out[23]:
```
3
```

In [24]:
```
c.print() #====>2 + 3i
```

```
2 + 3 i
```

In [29]:
```
class Complex:
    def __init__(self,real=0,imag=0) :
        self.real=real
        self.imag=imag
    def print(self):
        print(self.real, "+",self.imag,"i")
    def add(self,doosra_com):
        self.real=self.real + doosra_com.real
        self.imag=self.imag + doosra_com.imag

c1=Complex(2,3)

c2=Complex(4,5)
c1.add(c2) # c1=c1+c2
```

In [30]:
```
c1.print()
```

```
6 + 8 i
```

In [31]:
```
c3=Complex()  # 0 + 0 i
c4=Complex(2) # 2 + 0 i
c5=Complex(2,3) # 2 + 3 i ====>implemented
```

In [32]:
```
c3.print()
```

```
0 + 0 i
```

In [33]:
```
c4.print()
```

```
2 + 0 i
```

In [34]:
```
c5.print()
```

```
2 + 3 i
```

In [39]:
```
class Fraction:
    def __init__(self,num=0,deno=1):
        self.num=num
        self.deno=deno
    def print(self):
        print(self.num,"/",self.deno)
```

In [40]:
```
f1=Fraction(2,3) #2/3
```

In [41]:
```
f1.print()
```

```
2 / 3
```

In [42]:
```
f2=Fraction()
```

In [43]:
```
f2.print()
```

```
0 / 1
```

In [44]:
```
f3=Fraction(4)
```

In [45]:
```
f3.print()
```

```
4 / 1
```

In [47]:
```
class Fraction:
    def __init__(self,num=0,deno=1):
        self.num=num
        self.deno=deno
    def print(self):
        print(self.num,"/",self.deno)
    def normalize(self):
        min_value=min(self.num,self.deno)
        while min_value>1:
            if self.num%min_value==0 and self.deno%min_value==0 :
                break
            min_value=min_value-1
        self.num=self.num//min_value
        self.deno=self.deno//min_value
```

In [48]:
```
f=Fraction(12,8) # 12 / 8 ===> 3 / 2
```

In [49]:
```
f.print()
```

```
12 / 8
```

In [50]:
```
f.normalize()
```

In [52]:
```
f.print()
```

```
3 / 2
```

In [53]:
```
class Fraction:
    def __init__(self,num=0,deno=1):
        self.num=num
        self.deno=deno
    def print(self):
        print(self.num,"/",self.deno)
    def normalize(self):
        min_value=min(self.num,self.deno)
        while min_value>1:
            if self.num%min_value==0 and self.deno%min_value==0 :
                break
            min_value=min_value-1
        self.num=self.num//min_value
        self.deno=self.deno//min_value
    def subtract(self,doosra):
        up_num=self.num * doosra.deno - self.deno *doosra.num
        up_deno=self.deno * doosra.deno
        self.num=up_num
        self.deno=up_deno
        self.normalize()
```

In [54]:
```
f1=Fraction(1,2)
```

In [55]:
```
f2=Fraction(3,4)
```

In [56]:
```
f1.subtract(f2)
```

In [57]:
```
f1.print()
```

```
1 / -4
```

In [ ]:
```

```

THANK - YOU