Data Science & AI
&
NIC - Param

Python-For Data Science

OOPs

Lecture No.- 05

By- Pankaj Sharma Sir

Recap of Previous Lecture

Topic Object-Oriented Programming Part -04

Topics to be Covered

Topic Object-Oriented Programming Part -05

Topic : Object-Oriented Programming

Python → object

Class abc :

Every class inherited from Object class

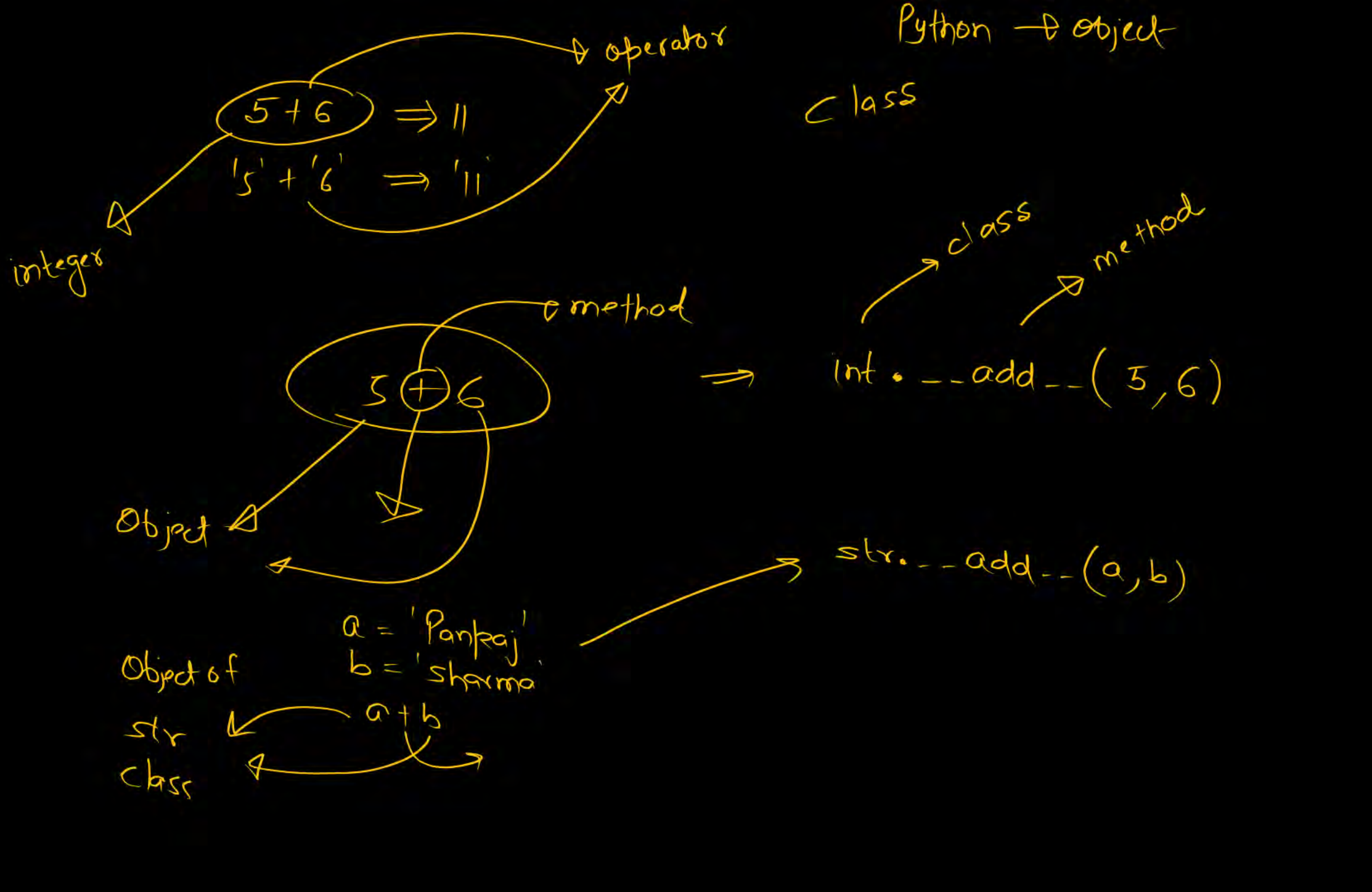
Class abc(object) :

--new--

--str--

--init--

{ Over-riding
  method overloading }


operator
5 + 6 ⇒ 11
'5' + '6' ⇒ '11'
integer
Python → object-
class
method
5 + 6
Object
⇒
class
method
int . __add__ (5, 6)
str . __add__ (a, b)
a = 'Pankaj'
b = 'sharma'
Object of
str
class
a + b

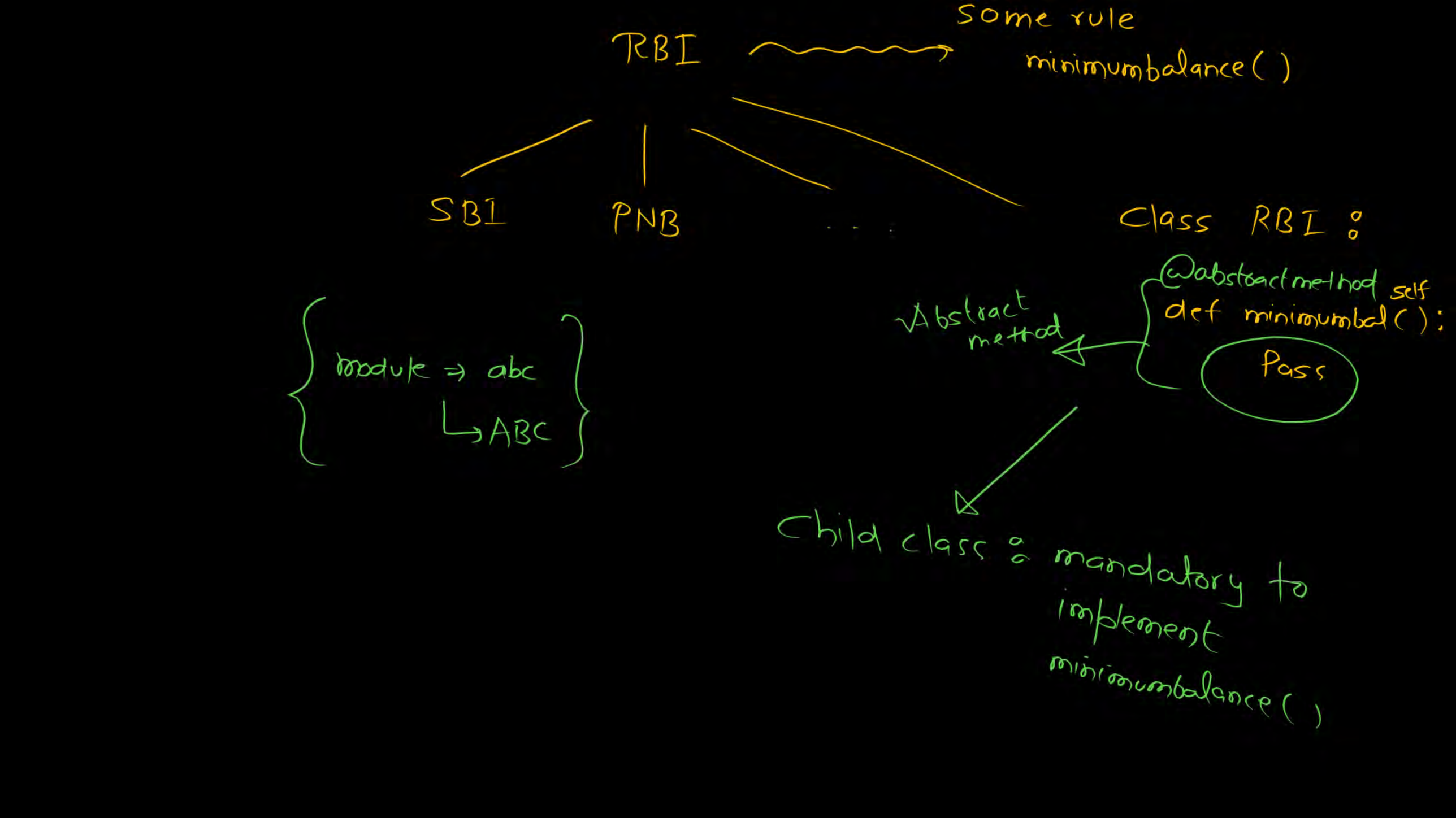
class complex:

```
c1 = complex(2,3)
c2 = complex(5,6)
```


```
def --add--(self, doosra):

    new_real = self.real + doosra.real
    new_imag = self.imag + doosra.imag
    c3 = Complex(new_real, new_imag)
    return c3
```

- Abstract class
- Abstract method


RBI
some rule
minimumbalance()
SBI
PNB
. . .
Class RBI :
@abstractmethod self
def minimumbal():
Pass
Abstract method
module ⇒ abc
↳ ABC
Child class : mandatory to implement minimumbalance()

var	method
3	3
1) Local	1) instance
2) instance	2) class method
3) Class level (static)	3) static method

Inheritance

✓ Encapsulation

Polymorphism :

method overriding

abstract method

abstract class

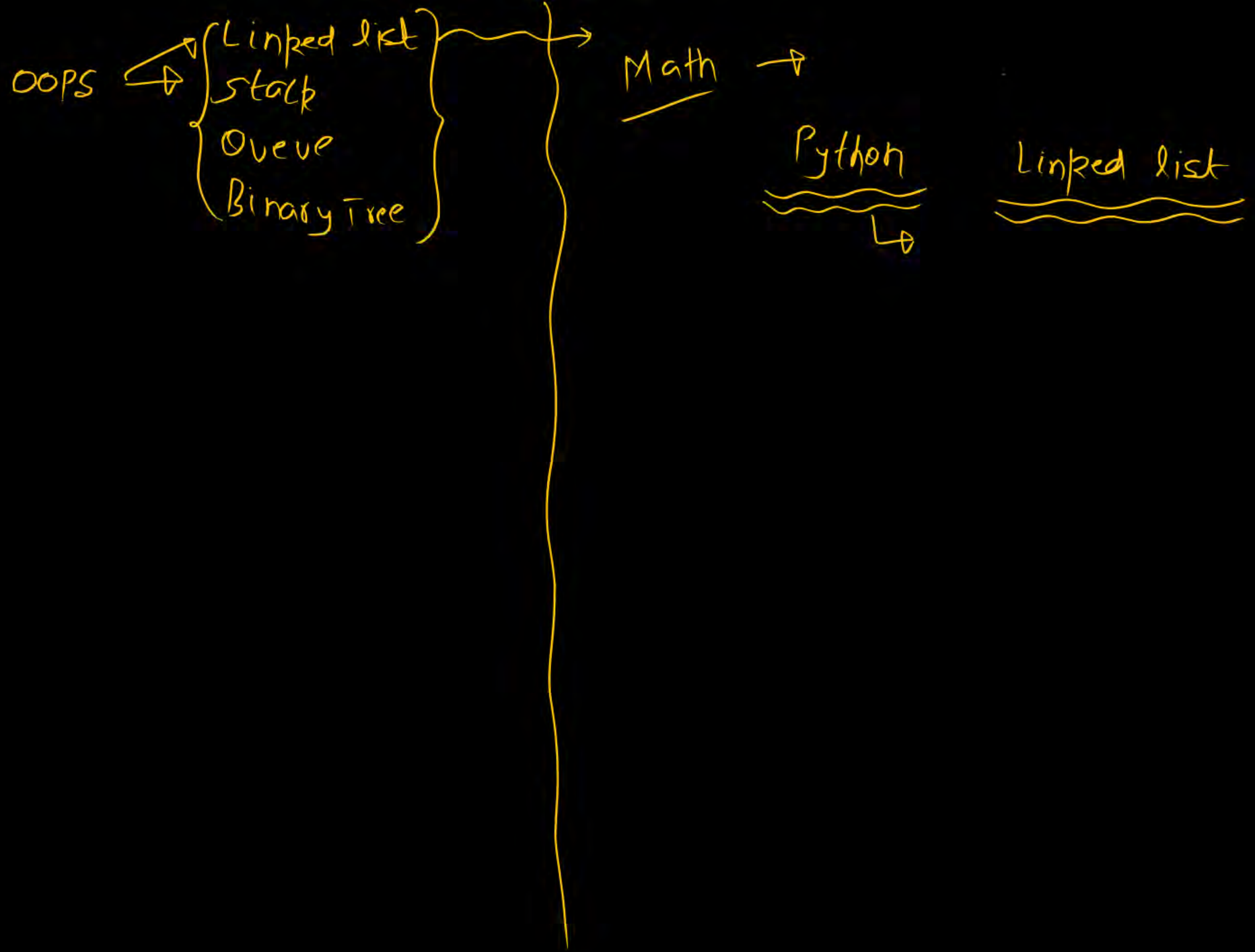
operator overloading

class

Object

refrence variable

* Public, private


OOPS
Linked list
Stack
Queue
Binary Tree
Math
Python
Linked list

In [1]:
```
5+6
```

Out[1]:
```
11
```

In [2]:
```
int.__add__(5,6)
```

Out[2]:
```
11
```

In [3]:
```
str.__add__('pankaj','sharma')
```

Out[3]:
```
'pankajsharma'
```

In [4]:
```
float.__add__(12.3,45.7)
```

Out[4]:
```
58.0
```

In [8]:
```
list.__add__([1,2,3],[5,6,7])
```

Out[8]:
```
[1, 2, 3, 5, 6, 7]
```

In [30]:
```
class Complex:
    def __init__(self,real,imag):
        self.real=real
        self.imag=imag
    def __add__(self,doosra):
        new_real=self.real + doosra.real
        new_imag=self.imag + doosra.imag
        c3=Complex(new_real,new_imag)
        return c3
    def print(self):
        print(self.real,"+",self.imag,"i")
    def __str__(self):
        return str(self.real)+ "+" + str(self.imag)+ "i"
c1=Complex(2,3)
```

In [31]:
```
c1.print()
```

```
2 + 3 i
```

In [32]:
```
c2=Complex(3,6)
c2.print()
```

```
3 + 6 i
```

In [33]:
```
c=c1 + c2
```

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

```
5 + 9 i
```

In [35]:
```
print(c)  #object class ka __str__ call hota hai
```

```
5+9i
```

In [39]:
```
from abc import ABC,abstractmethod
class RBI(ABC):
```

```
    def __init__(self):
        print("RBI")
    @abstractmethod
    def minbalance(self):
        pass
    @abstractmethod
    def f(self):
        pass
```

In [40]:
```
c=RBI()  #abstract class ka object ni banta
#any class having atleast 1 abstarct method ==>abstract class
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[40], line 1
----> 1 c=RBI()

TypeError: Can't instantiate abstract class RBI with abstract methods f, minbalance
```

In [41]:
```
from abc import ABC,abstractmethod
class RBI(ABC):
    def __init__(self):
        print("RBI")
    @abstractmethod
    def minbalance(self):
        pass

    def f(self):
        pass
r=RBI()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[41], line 11
      9     def f(self):
     10         pass
---> 11 r=RBI()

TypeError: Can't instantiate abstract class RBI with abstract method minbalance
```

In [42]:
```
from abc import ABC,abstractmethod
class RBI(ABC):
    def __init__(self):
        print("RBI")

    def minbalance(self):
        pass

    def f(self):
        pass
s=RBI()   #RBI is not a abstract class
```

```
RBI
```

In [44]:
```
from abc import ABC,abstractmethod
class RBI(ABC):
    def __init__(self):
        print("RBI")
    @abstractmethod
    def minbalance(self):
```

```
        pass
    @abstractmethod
    def f(self):
        pass
class ICICI(RBI):
    def __init__(self):
        print("ICICI")
    def minbalance(self):
        print("implemented icici rule")
    def f(self):
        print("hello")
    def g(self):
        print("hello g")
c=ICICI()
```

```
ICICI
```

In [45]:
```
#Exception and Error
a=10 #no problem
```

In [46]:
```
a="xyz" #no problem
```

In [47]:
```
a=b    #b does not exist , b===>name does not exist for Python
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[47], line 1
----> 1 a=b

NameError: name 'b' is not defined
```

In [48]:
```
a=23/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[48], line 1
----> 1 a=23/0

ZeroDivisionError: division by zero
```

In [49]:
```
d='pankaj' + 6
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[49], line 1
----> 1 d='pankaj' + 6

TypeError: can only concatenate str (not "int") to str
```

In [50]:
```
a=int(input())
b=int(input())
a/b
```

```
abc
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[50], line 1
----> 1 a=int(input())
      2 b=int(input())
      3 a/b

ValueError: invalid literal for int() with base 10: 'abc'
```

In [51]:
```
#try except
a=int(input("Enter the numerator"))
b=int(input("enter denominator"))
ans=a/b
print(ans)
```

```
Enter the numerator12
enter denominator4
3.0
```

In [52]:
```
#try except
a=int(input("Enter the numerator"))
b=int(input("enter denominator"))
ans=a/b
print(ans)
```

```
Enter the numeratorabc
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[52], line 2
      1 #try except
----> 2 a=int(input("Enter the numerator"))
      3 b=int(input("enter denominator"))
      4 ans=a/b

ValueError: invalid literal for int() with base 10: 'abc'
```

In [53]:
```
#try except
a=int(input("Enter the numerator"))
b=int(input("enter denominator"))
ans=a/b
print(ans)
```

```
Enter the numerator12
enter denominatora
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[53], line 3
      1 #try except
      2 a=int(input("Enter the numerator"))
----> 3 b=int(input("enter denominator"))
      4 ans=a/b
      5 print(ans)

ValueError: invalid literal for int() with base 10: 'a'
```

In [54]:
```
#try except
a=int(input("Enter the numerator"))
b=int(input("enter denominator"))
ans=a/b
print(ans)
```

```
Enter the numerator12
enter denominator0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[54], line 4
      2 a=int(input("Enter the numerator"))
      3 b=int(input("enter denominator"))
----> 4 ans=a/b
      5 print(ans)

ZeroDivisionError: division by zero
```

In [55]:
```
try:
    a=int(input("Enter the numerator"))
    b=int(input("enter denominator"))
    ans=a/b
    print(ans)
except ValueError:
    print("Nemerator and denominator must be number type")
```

```
Enter the numeratorabc
wrong
```

In [56]:
```
try:
    a=int(input("Enter the numerator"))
    b=int(input("enter denominator"))
    ans=a/b
    print(ans)
except ValueError:
    print("Nemerator and denominator must be number type")
```

```
Enter the numerator10
enter denominatorxyz
Nemerator and denominator must be number type
```

In [57]:
```
try:
    a=int(input("Enter the numerator"))
    b=int(input("enter denominator"))
    ans=a/b
    print(ans)
except ValueError:
    print("Nemerator and denominator must be number type")
```

```
Enter the numerator12
enter denominator0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[57], line 4
      2     a=int(input("Enter the numerator"))
      3     b=int(input("enter denominator"))
----> 4     ans=a/b
      5     print(ans)
      6 except ValueError:

ZeroDivisionError: division by zero
```

In [58]:
```
try:
    a=int(input("Enter the numerator"))
    b=int(input("enter denominator"))
    ans=a/b
```

```
    print(ans)
except ValueError:
    print("Nemerator and denominator must be number type")
except ZeroDivisionError :
    print("deno must be non zer")
```

```
Enter the numerator12
enter denominatorabc
Nemerator and denominator must be number type
```

In [60]:
```
try:
    a=int(input("Enter the numerator"))
    b=int(input("enter denominator"))
    ans=a/b
    print(ans)
except ValueError:
    print("Nemerator and denominator must be number type")
except ZeroDivisionError :
    print("deno must be non zer")
```

```
Enter the numerator10
enter denominator0
deno must be non zer
```

In [1]:
```
while 1:
    try:
        a=int(input("Enter the numerator"))
        b=int(input("enter denominator"))
        ans=a/b
        print(ans)
        break
    except ValueError:
        print("Nemerator and denominator must be number type")
    except ZeroDivisionError :
        print("deno must be non zero")
```

```
Enter the numerator12
enter denominator4
3.0
```

In [2]:
```
while 1:
    try:
        a=int(input("Enter the numerator"))
        b=int(input("enter denominator"))
        ans=a/b
        print(ans)
        break
    except (ValueError,ZeroDivisionError):
        print("Nemerator and denominator must be number type")
```

```
Enter the numerator12
enter denominatorxyz
Nemerator and denominator must be number type
Enter the numerator12
enter denominator0
Nemerator and denominator must be number type
Enter the numerator12
enter denominator3
4.0
```

In [3]:
```
while 1:
    try:
        a=int(input("Enter the numerator"))
        b=int(input("enter denominator"))
        ans=a/b
        print(ans)
        break
    except ValueError:
        print("Nemerator and denominator must be number type")
    except ZeroDivisionError :
        print("deno must be non zero")
```

```
Enter the numerator12
enter denominator3+4j
Nemerator and denominator must be number type
Enter the numerator12
enter denominatorz
Nemerator and denominator must be number type
Enter the numerator[1,2,]
Nemerator and denominator must be number type
Enter the numerator12
enter denominator3
4.0
```

In [ ]:
```
#custom Exception ==>own exception
#u must know how to raise an exception
while 1:
    try:
        a=int(input("Enter the numerator"))
        b=int(input("enter denominator"))
        if b==0:
            raise ZeroDivisionError
        ans=a/b
        print(ans)
        break
    except ValueError:
        print("Nemerator and denominator must be number type")
    except ZeroDivisionError :
        print("deno must be non zero")
```

```
Enter the numerator12
enter denominator0
deno must be non zero
```

In [1]:
```
while 1:
    try:
        a=int(input("Enter the numerator"))
        b=int(input("enter denominator"))
        if b==0:
            raise TypeError
        ans=a/b
        print(ans)
        break
    except ValueError:
        print("Nemerator and denominator must be number type")
    except ZeroDivisionError :
        print("deno must be non zero")
```

```
Enter the numerator12
enter denominator0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[1], line 6
      4 b=int(input("enter denominator"))
      5 if b==0:
----> 6     raise TypeError
      7 ans=a/b
      8 print(ans)

TypeError:
```

In [6]:
```
class Zerodeno(Exception):
    "error"
    pass
while 1:
    try:
        a=int(input("Enter the numerator"))
        b=int(input("enter denominator"))
        if b==0:
            raise Zerodeno('Hello')
        ans=a/b
        print(ans)
        break
    except ValueError:
        print("Nemerator and denominator must be number type")
    except ZeroDivisionError :
        print("deno must be non zero")
```

```
Enter the numerator12
enter denominator0
---------------------------------------------------------------------------
Zerodeno                                  Traceback (most recent call last)
Cell In[6], line 9
      7 b=int(input("enter denominator"))
      8 if b==0:
----> 9     raise Zerodeno('Hello')
     10 ans=a/b
     11 print(ans)

Zerodeno: Hello
```

In [ ]:

THANK - YOU