Principles of Object Oriented Programming (OOP's)

paradigm

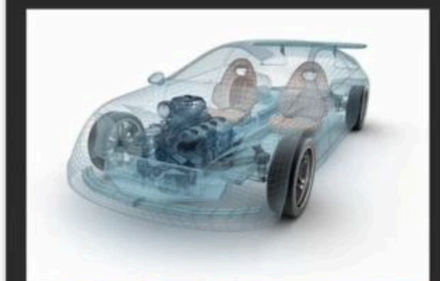
Student
Car
WashingMachine
movie
item
Order

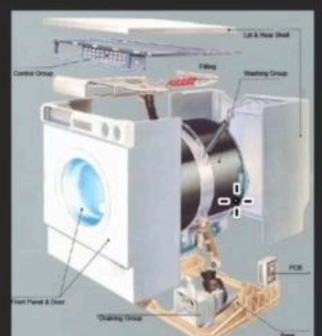
→ properties
→ Methods

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Encapsulation

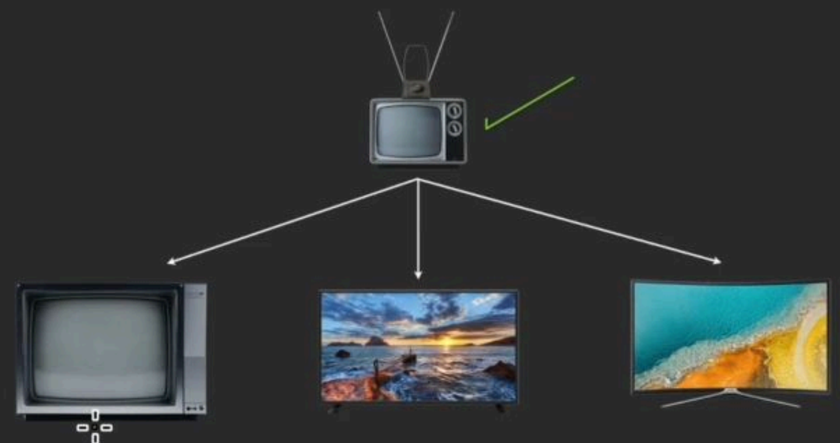
Abstraction



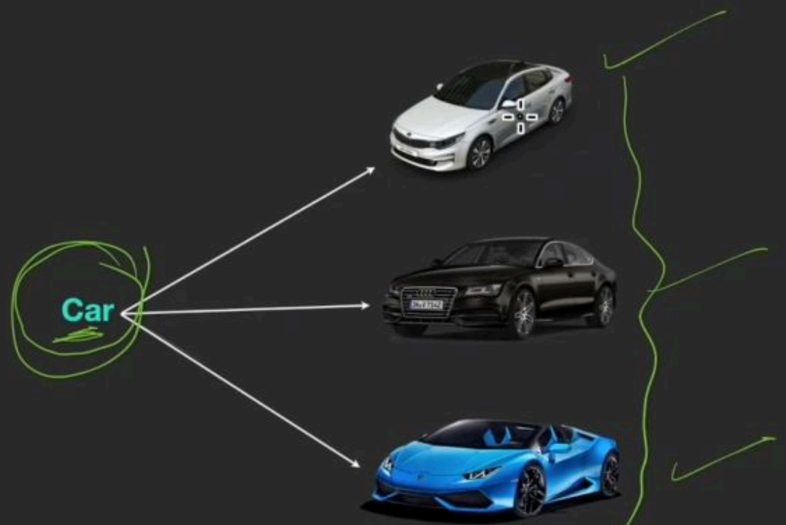


Inheritance

Inheritance



Polymorphism



Classes vs Objects

class cuboid

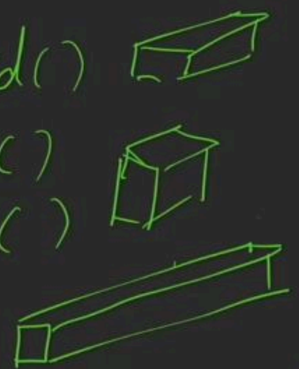
properties:
- length
- breadth
- height

methods:
- lidarea()
- totalarea()
- volume()

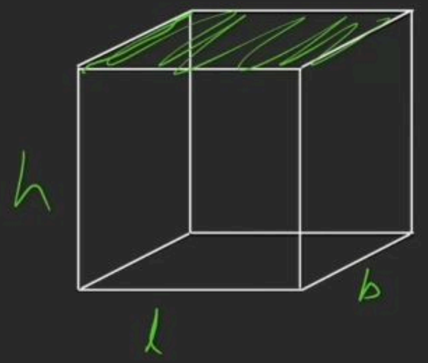
c1 = Cuboid()

c2 = Cuboid()

c3 = Cuboid()



cuboid



lidarea()

volume()

Python Course

class TV

properties:

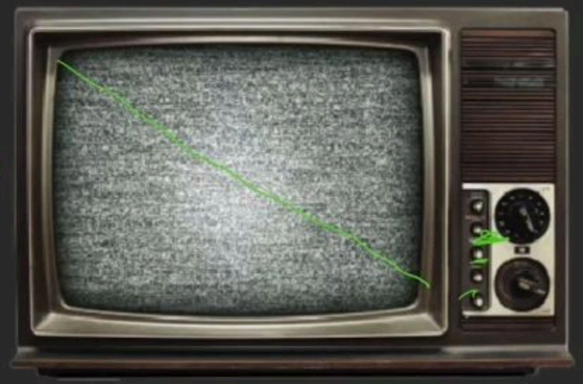
- type
- size
- company
- NoofChannel

methods:

- switchOn()
- " off()
- changeChannel()
- changeVolume()

tv1 = TV()

tv2 = TV()



object

Python Course

class student

properties
- rollno
- name
- address
- standard
- section
- mark1
- mark2
- :

methods :
- attend()
- exam()
- result()

blueprint

```
s1 = student()
s2 = student()
```

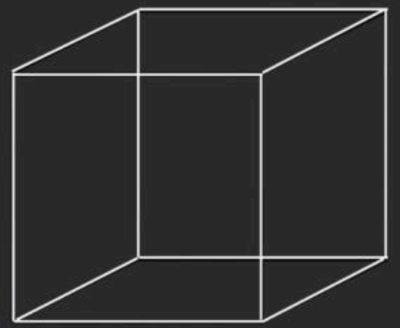
physical

object

Python Course • 2 Viewing

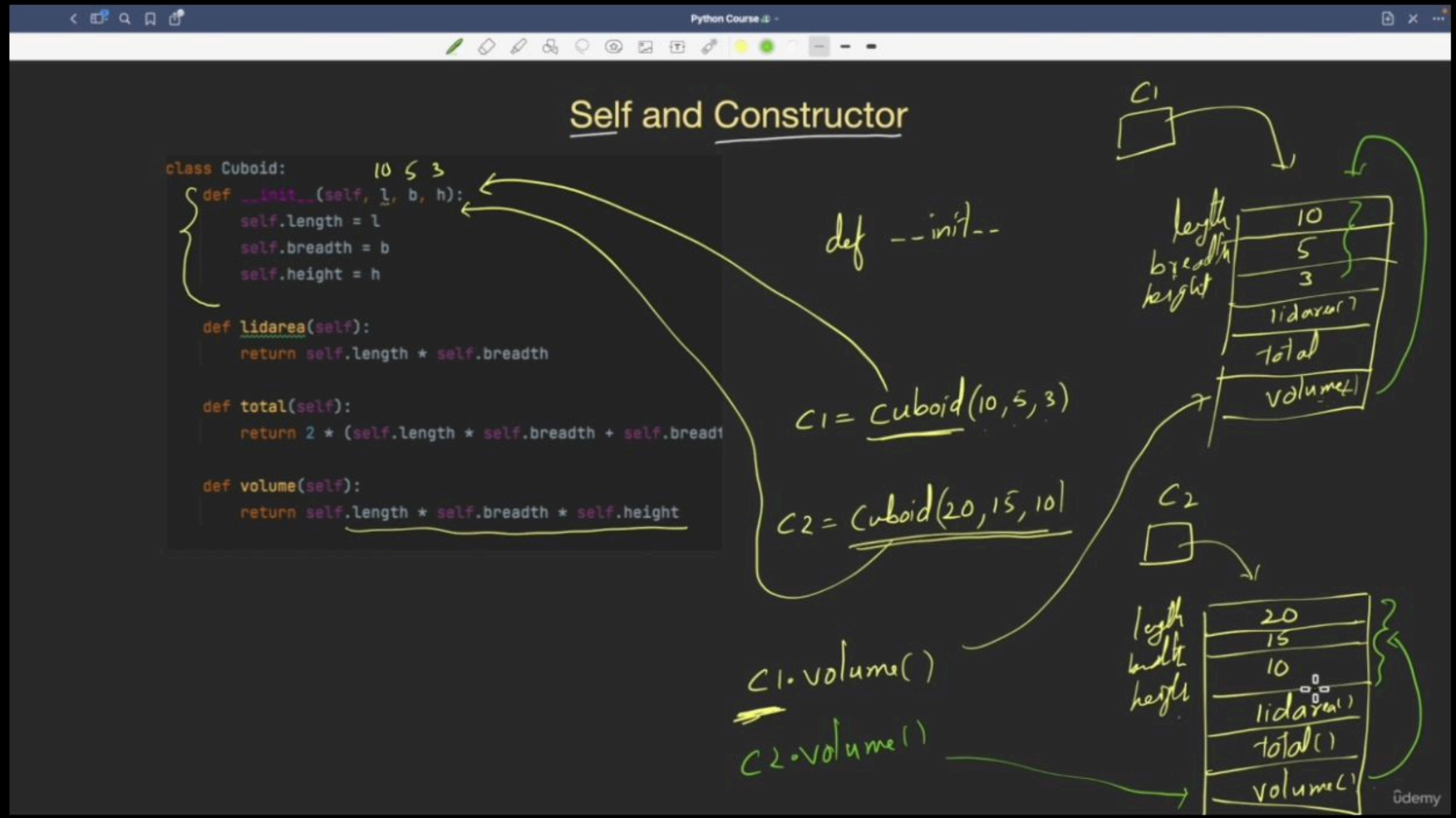
```
class cuboid
{
    int length;
    int breadth;
    int height;

    cuboid(int l, int b, int h)
    {
        length = l
        breadth = b
        height = h
    }
    int lidarea() { ... }
    int total() { ... }
    int volume() { ... }
}
```

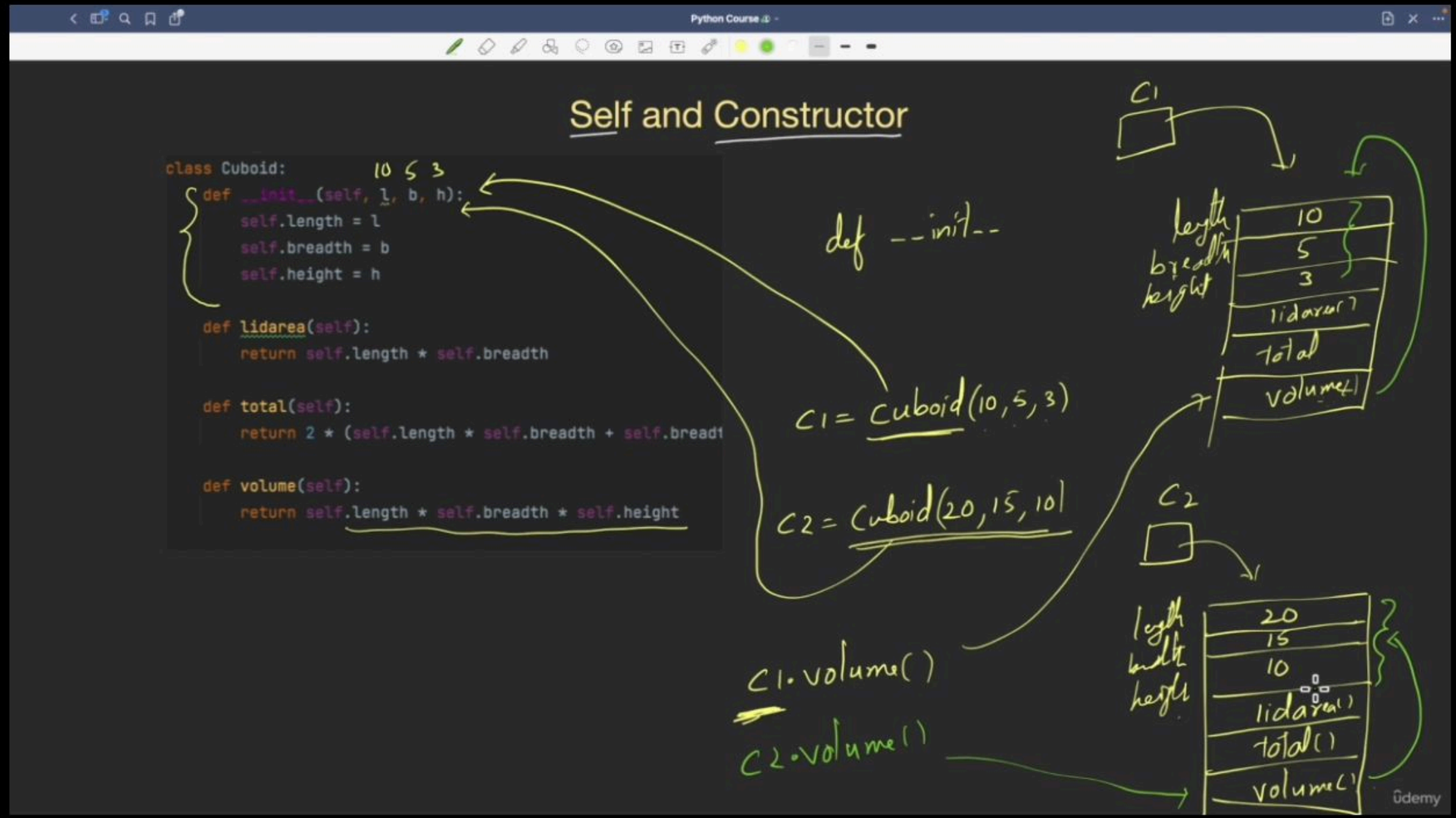
```
class cuboid:                    10 5 3
    def __init__(self, l, b, h)
        self.length = l
        self.breadth = b
        self.height = h

    def lidarea(self)
        return self.length * self.breadth

    def total(self)
        return 2*(self.length * self.breadth + ...)

    def volume(self)
        return self.length * self * breadth * self.height
```



Cuboid c1 = new Cuboid(10,5,3)

✗ c1 = cuboid(10,5,3)

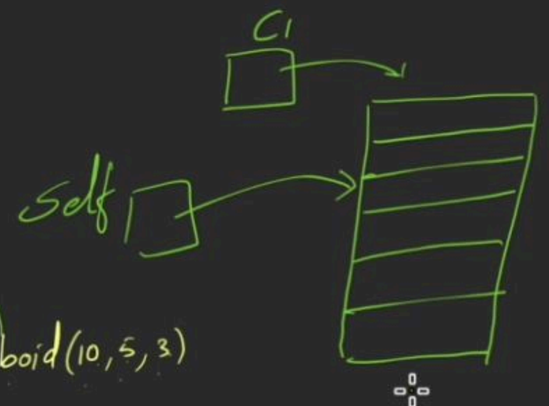

Python Course
Self and Constructor
class Cuboid:
def __init__(self, l, b, h):
self.length = l
self.breadth = b
self.height = h
def lidarea(self):
return self.length * self.breadth
def total(self):
return 2 * (self.length * self.breadth + self.breadt
def volume(self):
return self.length * self.breadth * self.height
10 5 3
def --init--
c1 = Cuboid(10,5,3)
c2 = Cuboid(20,15,10)
c1.volume()
c2.volume()
c1
length
breadth
height
10
5
3
lidarea()
total()
volume()
c2
20
15
10
lidarea()
total()
volume()
Udemy


Python Course
Self and Constructor
class Cuboid:
def __init__(self, l, b, h):
self.length = l
self.breadth = b
self.height = h
def lidarea(self):
return self.length * self.breadth
def total(self):
return 2 * (self.length * self.breadth + self.breadt
def volume(self):
return self.length * self.breadth * self.height
10 5 3
def --init--
c1 = Cuboid(10,5,3)
c2 = Cuboid(20,15,10)
c1.volume()
c2.volume()
c1
length
breadth
height
10
5
3
lidarea()
total()
volume()
c2
20
15
10
lidarea()
total()
volume()
Udemy

Self and Constructor

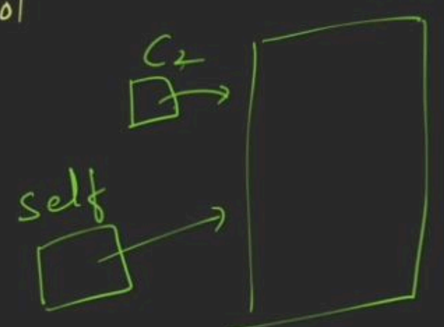
```
class Cuboid:
    def __init__(self, l, b, h):
        self.length = l
        self.breadth = b
        self.height = h

    def lidarea(self):
        return self.length * self.breadth

    def total(self):
        return 2 * (self.length * self.breadth + self.breadt

    def volume(self):
        return self.length * self.breadth * self.height
```

c1 10 5 3



c1 = Cuboid(10, 5, 3)

c2 = Cuboid(20, 15, 10)



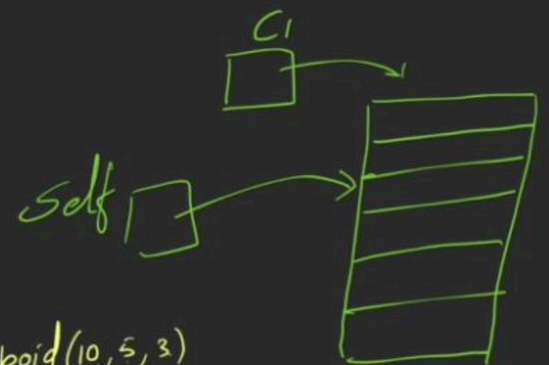
Self and Constructor

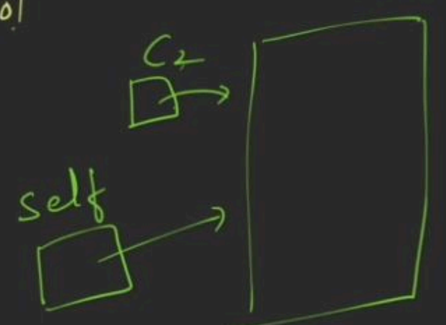
```
class Cuboid:
    def __init__(self, l, b, h):
        self.length = l
        self.breadth = b
        self.height = h

    def lidarea(self):
        return self.length * self.breadth

    def total(self):
        return 2 * (self.length * self.breadth + self.breadt

    def volume(self):
        return self.length * self.breadth * self.height
```

c2 20 15 10

c1 = Cuboid(10, 5, 3)

c2 = Cuboid(20, 15, 10)





Self and Constructor

```
class Cuboid:
    def __init__(self, l, b, h):
        self.length = l
        self.breadth = b
        self.height = h

    def lidarea(self):
        return self.length * self.breadth

    def total(self):
        return 2 * (self.length * self.breadth + self.breadt

    def volume(self):
        return self.length * self.breadth * self.height
```

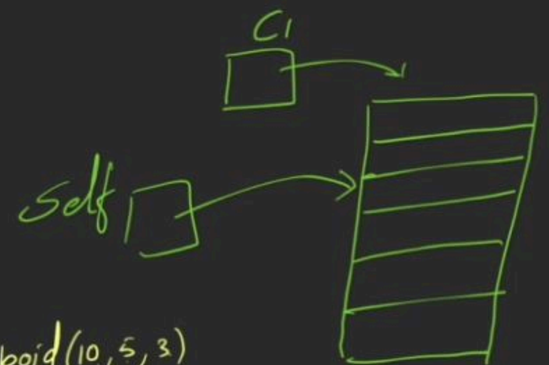
c2 20 15 10

c1

c1 = Cuboid(10, 5, 3)

c2 = Cuboid(20, 15, 10)

c1.volume()



Types Of Variables & Method

variables

- instance variables
- static/class variables

Methods

- Instance methods
- class methods
- static methods

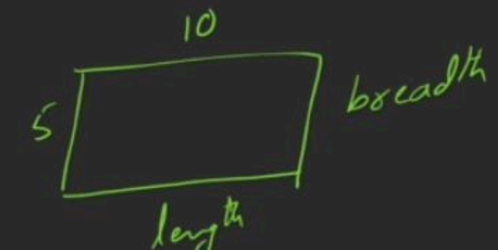
Instance Variable & Method

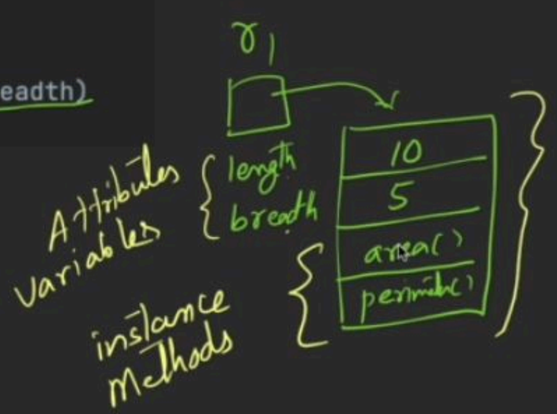
```
class Rectangle:
    def __init__(self,length, breadth):
        self.length=length
        self.breadth=breadth

    def area(self):
        return self.length * self.breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)
```

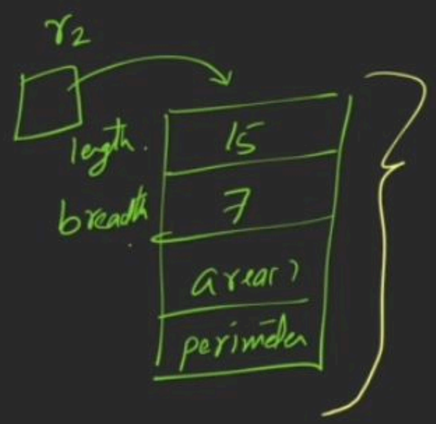
r1 10 5

r1 = Rectangle(10, 5)

r2 = Rectangle(15, 7)







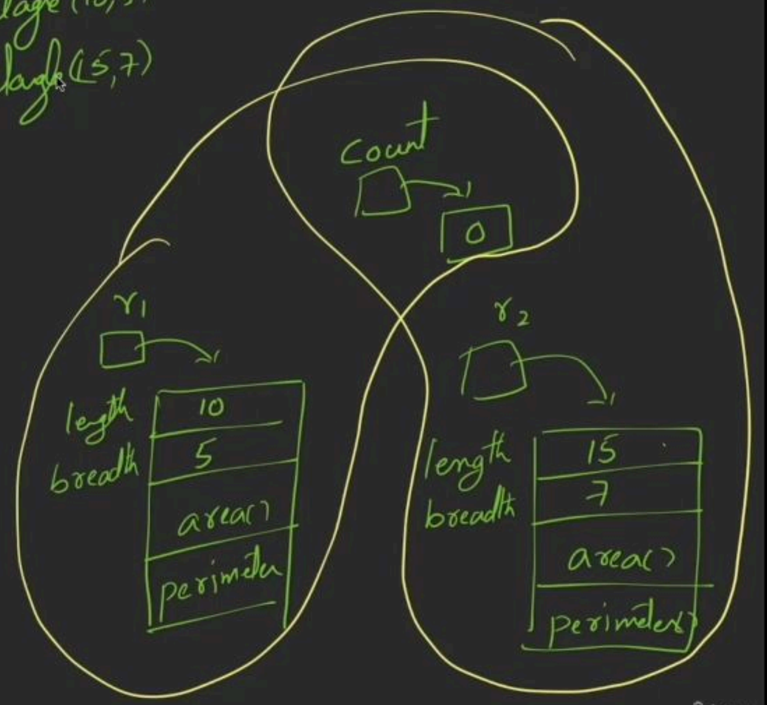
Class/Static Variable & Class Method

```
class Rectangle:
    count = 0
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth
        Rectangle.count += 1

    def perimeter(self):
        return 2 * (self.length + self.breadth)

    def area(self):
        return self.length * self.breadth
```

```
@classmethod
def countRect(cls):
    print(cls.count)
```

class Variable Rectangle

r1 = Rectangle(10,5)
r2 = Rectangle(15,7)



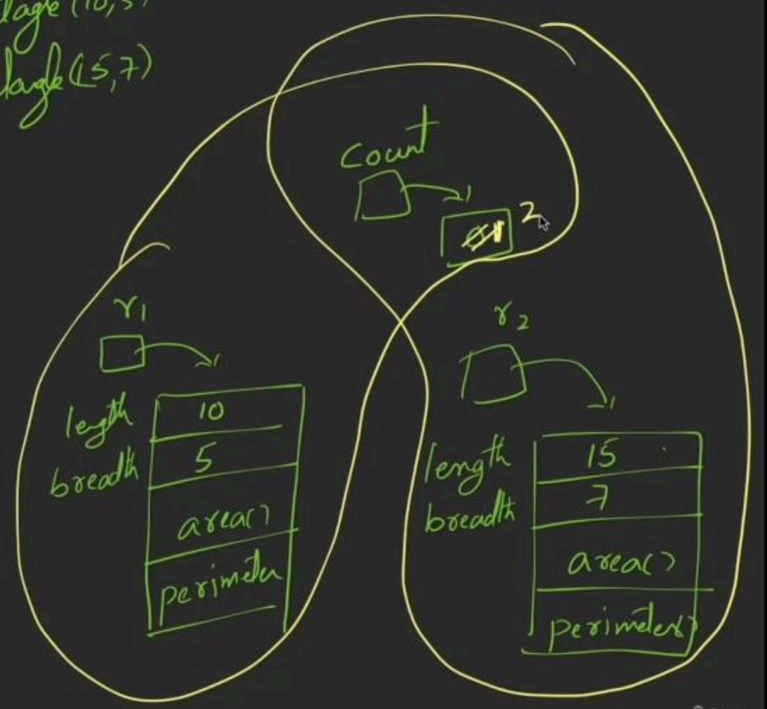
Class/Static Variable & Class Method

```
class Rectangle:
    count = 0
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth
        Rectangle.count += 1

    def perimeter(self):
        return 2 * (self.length + self.breadth)

    def area(self):
        return self.length * self.breadth
```

```
@classmethod
def countRect(cls):
    print(cls.count)
```

class Variable Rectangle

r1 = Rectangle(10,5)
r2 = Rectangle(15,7)



Static Methods

```python
class Rectangle:

    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)

    def area(self):
        return self.length * self.breadth
```

@staticmethod
def issquare(len, bre)
    return len == bre

@staticmethod

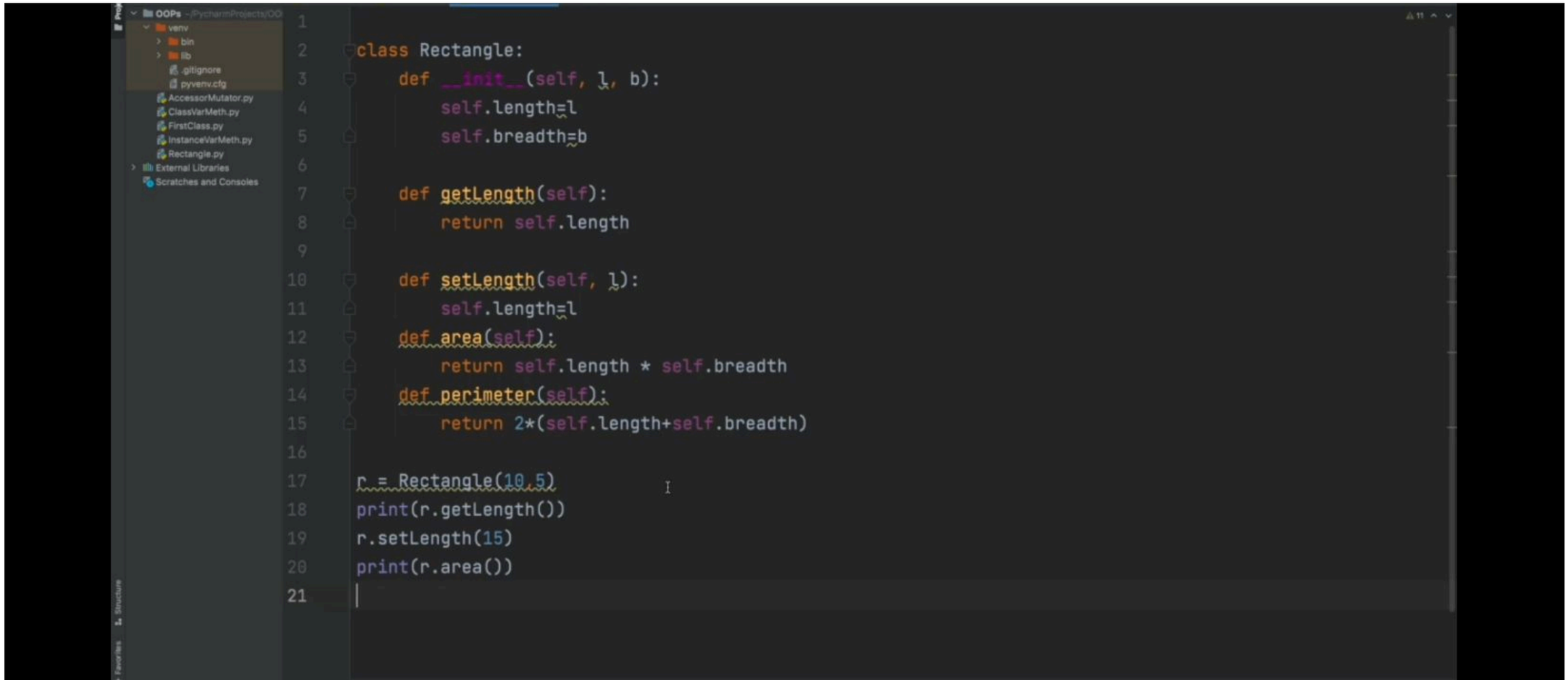
Accessors and Mutators

```
class Rectangle:

    def __init__(self, l, b):
        self.length = l
        self.breadth = b

    def getLength(self):
        return self.length

    def setLength(self, l):
        self.length = l
```

getBreadth(_)

setBreadth(__)


```
r = Rectangle(10,5)
print(r.getLength())
r.setLength(15)
```


OOPs ~/PycharmProjects/OO
venv
bin
lib
.gitignore
pyvenv.cfg
AccessorMutator.py
ClassVarMeth.py
FirstClass.py
InstanceVarMeth.py
Rectangle.py
External Libraries
Scratches and Consoles
class Rectangle:
def __init__(self, l, b):
self.length=l
self.breadth=b
def getLength(self):
return self.length
def setLength(self, l):
self.length=l
def area(self):
return self.length * self.breadth
def perimeter(self):
return 2*(self.length+self.breadth)
r = Rectangle(10,5)
print(r.getLength())
r.setLength(15)
print(r.area())
Structure
Favorites

Principles of object oriented programming(oop's)

It is a paradigm(oop's is a paradigm)

What is object?

- Everything in the world is tangible object or intangible object>
- Ex: student, car, washing machine, movie, order
- So we believe that anything in the world can be object.
- By defining in terms of its properties and methods

Encapsulation:

- means combining all the related things together
- Ex: car, Tv, Washing machine.

Abstraction:

- means showing required features and hiding internal details.
- To make an object first we design the classes
- Class: a Class is a blueprint of an object

Inheritance:

- inheriting the features of existing class.
- Ex: parent->child.

Polymorphism:

- one name and different actions

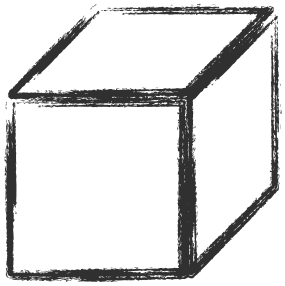
- By using a single item we can refer multiple things together.

Class v/s Object

- In OOP's we consider everything as an object and for every object there is a class
- So, class is a definition of an object . It is also know as blueprint , design , plan etc…
- Every object will have 2 things

1) Properties / Attributes
2) Methods / Operations / Function

Ex :



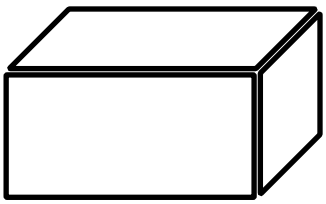
- The property of a cuboid are l , b , h
- The methods of a cuboid are lidarea( ) , volume ( ) , totalarea( )
- A sudo code for cuboid is

Class cuboid

properties :
Length
breathe
height

Methods :
lidarea( )
totalarea( )
volume ( )

How to Write a class



Methods :

lidarea( ) = l *b
total( ) = l * b * b * h * l * h
volume = l*b*h

Writing a class using java

Class cuboid
{
int length;
int breath;
int height;
cubiod(int length,int breath,int height)

#constructor will have same name as class name . It is use to initialise the object

{
length = l;
breadth = b;
height = h;
}
int lidarea( ){….}
int total( ){…..}
int volume ( ) {…..}
}

Writing a class in python

- When we write a variable we also need to initialise it . Initialisation is done in constructor (java) but in python we can write constructor method
- Initialising and declaring both should be done in python

```python
class Cuboid:
    def __init__(self, l, b, h):
        self.length = l
        self.breadth = b
        self.height = h

    def lidarea(self):
        return self.length * self.breadth

    def total(self):
        return 2 * (self.length * self.breadth + self.breadth * self.height + self.leng

    def volume(self):
        return self.length * self.breadth * self.height

c1 = Cuboid(10, 5, 3)
print(c1.volume())

c2 = Cuboid(20, 10, 5)
print(c2.volume())
```

- First parameter should be self , it is defined to declare and initialise it and properties are defined inside the function __init__ and this is standard function that acts as the constructor in class .

Create object in java’

Cuboid c1=new cuboid(10,5,3)

Create object in python

c1=cuboid(10,5,3)

Self and Construct

Self

- Self is the **reference** to the current value
- It is the first argument of the constructor
- If self is not given PVM will take the 1st parameter value as self by default
- You can write different names for self but self is recommended
- When you create 2 object the self will refer to the current object that is being called I.e, the self value will be same for these 2 object but it will show values of the method being called (i.e, 1st or 2nd)
- You can verify this by checking the ID of self in both method call

```python
class Cuboid:
    def __init__(self, l, b, h):
        print(id(self))
        self.length = l
        self.breadth = b
        self.height = h

    def lidarea(self):
        return self.length * self.breadth

    def total(self):
        return 2 * (self.length * self.breadth + self.breadth * self.height + self.lengt

    def volume(self):
        return self.length * self.breadth * self.height

c1 = Cuboid(10, 5, 3)
print(id(c1))
```

```
/Users/[redacted]/PycharmProject
140228827119568
140228827119568
```

Constructor

- In a method if _ _init_ _ is used then that method becomes constructor method as _ _init_ _ is a constructor.
- When you create an object of this class then automatically constructor method is called and initialised
- If a class having attributes then its mandatory to have _ _init_ _ method. Therefore every class must have _ _init_ _
- When _ _init_ _ is not given in a class then PVM will provide default constructor which will not have any property
- You can write more than 1 constructor but only 1 will execute but its not recommended to do so
- You can give default argument in parameter and when you call the method you can either give values or it will consider the default values i.e you can make constructor using different Parameters
- Constructors are not overloaded

```python
class Cuboid:
    def __init__(self, l, b, h):
        print(id(self))
        self.length = l
        self.breadth = b
        self.height = h

    def lidarea(self):
        return self.length * self.breadth

    def total(self):
        return 2 * (self.length * self.breadth + self.breadth * self.height + self.lengt

    def volume(self):
        print(id(self))
        return self.length * self.breadth * self.height

c1 = Cuboid(10, 5, 3)
print(id(c1))
c1.volume()
```

```
140496912371664
140496912371664
140496912371664
```

Program explanation(1 and 2):

- A class of cuboid is created , in method we are writing a constructor it takes 1st parameter as self, and we are initialisation it by writing its properties I.e length, breath and height , we are printing the ID of self as well to show that only one self is used for multiple methods call.
- Then we are defining a method for lidarea , total area and volume and initialisation it with self .
- When printing the results for lidarea , total area and volume we can see that the ID of self is constant while ans for theses 3 methods are different

Types of Variables and Methods

In python there are different types of variables and methods

Variables:

- Instance variable
- Static variable

Methods:

- Instance method
- Class method
- Static method

Instance :

```python
class Rectangle:
    def __init__(self,length, breadth):
        self.length=length
        self.breadth=breadth

    def area(self):
        return self.length * self.breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)
```

Example: Rectangle

Breadth

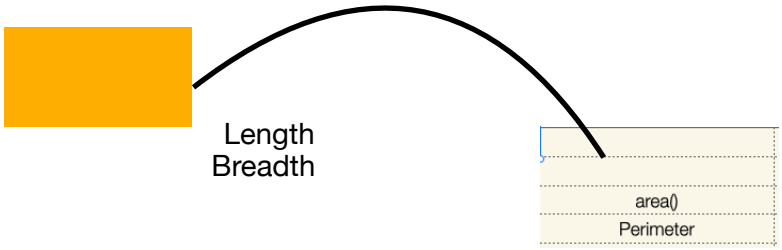
Length

area= length*breadth

Creating a variable for rectangle
r1= rectangle()# automatically calls the constructor of the class

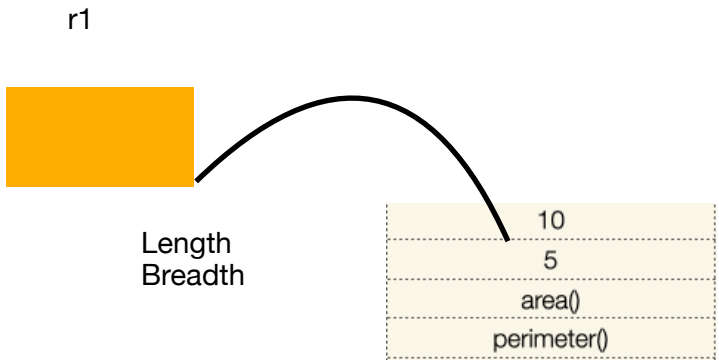
In the memory it looks like

r1



It will form like this

r1



r2= Rectangle(15,7) # another object

So again one more object screamed in memory

These are called instance of class

Each instance will have its own members

The parameter that is always taken by instance method is self and using self they access instance variable and the methods which are accessing them are called as instance methods.

Ways of creating instance variables.

So, there are multiple ways we can declare instance variable.

Program:

Input:

```
class Test:# creating a class
    def __init__(self):# constructor with instance variable
        self.a = 10
t1 = Test()
print(dir(t1))
```

Output:
['__doc__', '__init__', '__module__', ‘a']

So it is having one variable that is a

Program

Input:

```
class Test:# creating a class
    def __init__(self):# constructor with instance variable
        self.a = 10
        def fun(self):
            self.b=20
t1 = Test()
t1.c = 30
print(dir(t1))
```

Output:
['__doc__', '__init__', '__module__', 'a', 'c']

It is still having a and function but b is not there for that function should be called
We have added one more variable to it so, it means we can declare and initialise
Instance variable in init method also and other method also.

Instead of self we can use the variable name and.
Self is not a keyword it can be anything. The first we are writing is constructor is called as (instance variable)

We can give anytime instead of self

If a method is accessing instance variable then that method becomes instance method.

Class and Static Variable

```
class Rectangle:

    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)

    def area(self):
        return self.length * self.breadth
```

When we declare a variable inside a class but outside any method, it is called as class or static variable in python.

- self.length and breath are instance variable.

  def perimeter(self)

- Class variables can be accessed from within the class as well as from outside the class.

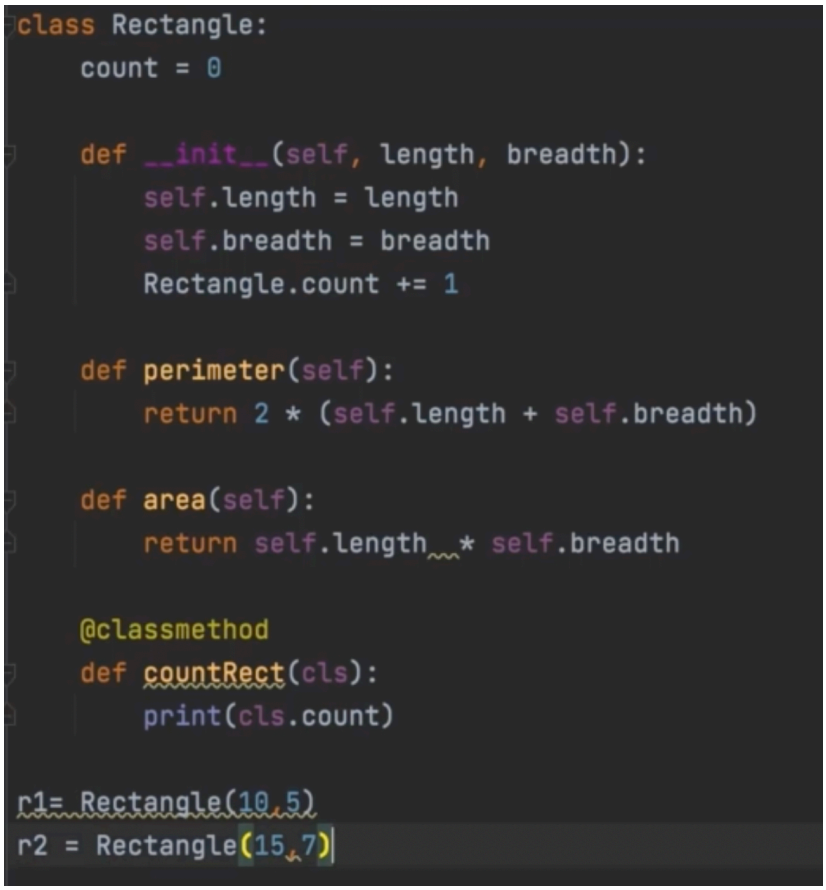
class and static method can access class variables. Each method has its own way of accessing static variables.

There are two ways of accessing class variables from inside class method-

- cls.ClassVariable
- ClassName.ClassVaraiable

There is only one way to access class variables from static method-

- ClassName.ClassVariable

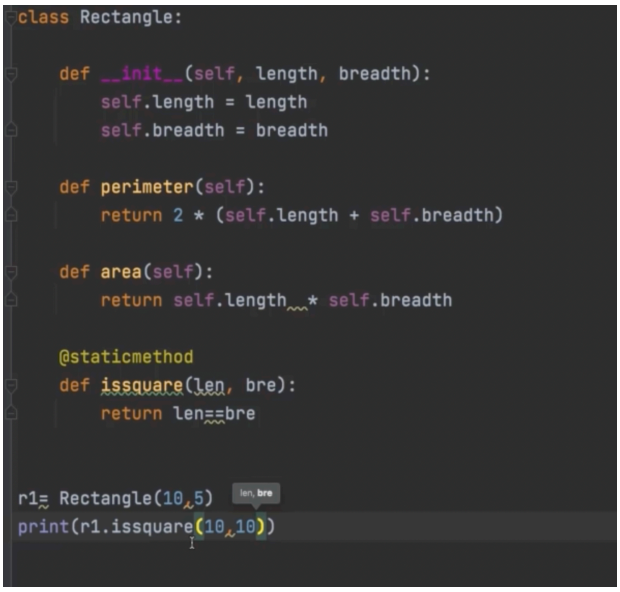


```python
class Rectangle:
    count = 0

    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth
        Rectangle.count += 1

    def perimeter(self):
        return 2 * (self.length + self.breadth)

    def area(self):
        return self.length * self.breadth

    @classmethod
    def countRect(cls):
        print(cls.count)

r1= Rectangle(10,5)
r2 = Rectangle(15,7)
```

- Here count= 0
- It is known as class or static cause its not inside the method
- The counter belongs to rectangle r1 and r2 that means count variable is common to both objects (r1 and r2)
- Count will belong to class (Rectangle) so it is also static(fixed, single copy to both r1,r2)
- @classmethod (decorator)

Static Methods

- Instance method access instance variable,class method access class variable.
- This methods are like global function.if there is any relation in the class we can put inside the class.
- The staticmethod() returns a static method for a function passed as the parameter.
-



```
class Rectangle:

    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)

    def area(self):
        return self.length * self.breadth

    @staticmethod
    def issquare(len, bre):
        return len==bre


r1= Rectangle(10,5)
print(r1.issquare(10,10))
```

- Example : Given rectangle is square or not
- If the length and breath both are equal then it is said as square
- This program will return true or false
- It is not accessing instance variable or class variable such methods are said as static method (its doesn't access)
- We can use the decorator @staticmethod . It is not compulsory . It will just help to read (readable)

Accessors and Mutators

These are the types of methods that can be written inside any class

Accessor methods are useful for reading the property of a class of an object.

Mutator is used for writing and updating properties of class and it’s objects.

So these methods can be called as reading and writing methods

These methods are followed by object oriented programming

Example : class Rectangle

Program:

Input:

```
class Rectangle:
    def __init__(self, l, b):
     self.length = l
     self.breadth = b

    def getlength(self):#get method for length
        return self.length#returning parameter length for changing the length

    def setlength(self, l):
        self.length=l

    def area(self):#calculating area of the rectangle
        return self.length * self.breadth

    def perimeter(self):# this will return perimeter of a rectangle
        return 2*(self.length+self.breadth)

r=Rectangle(10,20)
print(r.getlength())
r.setlength(15)#calculating the area by 15
print(r.area())#checking the area
```

Output:

```
10
300
```

Polymorphism

Polymorphism: many functions

Polymorphism means One name different actions

```
def Driver(car):
    car.drive()

class Creta:
    def drive(self):
        print('Creat is driving')

class Mercedes:
    def drive(self):
        print('Mercedes is driving')

c = Mercedes()
Driver(c)
```

Driver is a function who is driving the car and running cars.. so it is the example of polymorphism a driver can use or run multiple cars means different actions and forms but same function

```
def PetLover(pet):
    pet.talk()
    pet.walk()


class Duck:
    def talk(self):
        print('Duck is Talking')

    def walk(self):
        print('Duck is Walking')

class Dog:
    def talk(self):
        print('Dog is Talking')

    def walk(self):
        print('Dog is Walking')
```

taking method as pet lover and passing parameter as pet

#defining a class duck

#defining class dog

```
d = Dog()
PetLover(d)
```

Method Overloading

Method overloading is used for achieving polymorphism

What is polymorphism:

- One name different actions or multiple actions
- Python supports overloading writing one method which can act differently in different situations.

How python supports overloading ?

Class Arith:# creating class
Def sum(x, y)# takes two parameters
Return x+y # returning parameters by adding

a= arith()
print(a.sum(10,5))# calling sum
A.sum(8.5, 7.6)

print(a.sum("Hello", "world")

- It is adding int, float , string it is nothing but we are achieving polymorphism
- Python supports overloading implicitly.

Program:

Input:

```
class Arith:
    def sum(self,a,b):# takes two parameters
     return a+b
a= Arith()
print(a.sum(10, 5))
```

Output:

15

Input:

```
class Arith:
    def sum(self,a,b):# takes two parameters
     return a+b
a= Arith()
print(a.sum(8.5, 7.6))
```

Output:
16.1
Input:

```
class Arith:
    def sum(self,a,b):# takes two parameters
     return a+b
a= Arith()
print(a.sum('Hello  ', 'world'))
```

Output:

Hello World

- Concatenation means it will just attach first string to the second string

Input:

```
class Arith:
    def sum(self,a,b):# takes two parameters
     return a+b

    def sum(self, x, y, z):# takes three parameters
        return x + y + z

a= Arith()
print(a.sum('Hello  ', 'world'))
print(a.sum(5, 10, 3))# calling the sum
```

Output:

error

- Python does not allow two methods using same name.
- It will try to call out the second method and shadowed the second method
- We can never call the first method by calling two parameters

How single parameter works for both 2 and 3 parameters

Input:

```
class Arith:
    def sum(self, a, b, c=None):
        s = a + b
        if c==None:
            return s
        else:
            return s + c


a = Arith()
print(a.sum(5,10,3))

print(a.sum(8, 9))
```

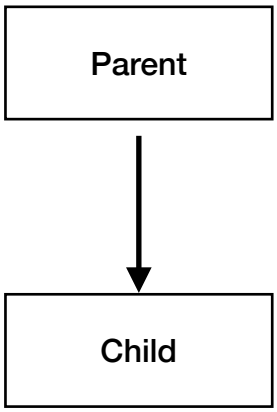
Output:

18
17

- So python is having method overloading implicitly
- We don’t have to write multiple methods with the same name but the single method can behave in different type of method.

Method Overriding

- Redefining the method of parent class in child class is said as method overriding
- In other words, the child class has access to the properties and functions of the parent class method while also extending additional functions of its own to the method.




```
Class iphone6:
def home(self)

Class iphone x:
```

- iphone x is newly designed phone . It is inheriting the features of iphone6 . but it is having new home method . it is not borrowing the method (iphone 6) as it is because they was not physical button

- Both have home method . But the child method will be called parent method will be shoddowed we cant all it directly.To override the Parent Class method, you have to create a function in the Child class with the same name and the same number of parameters.
- The child class should have the same name and the same number of parameters as the parent class.

- For self: to call a function firstly we have to create a variable then we can call the function

- Example : ph = iPhone(6)——-function calling
- ph.home() ——-method calling

- You can’t use super outside the class .method overriding cannot be performed in the same class, and overriding can only be executed when a child class is derived through inheritance.
- super() home() ——— this is how parent method is called . So it should be called inside the class

Operator Overloading

- Operator Overloading is where operators **behave differently** in different situations
- Operator Overloading is possible in normal DataType also
- Using operators we can **achieve polymorphism**
- Lets see this with an example
- We can add 2 rational numbers using operator overloading

```
class Rational:
    def __init__(self, p=1, q=1):
        self.p=p
        self.q=q

    def __add__(self, other):
        s = Rational()
        s.p = self.p * other.q + self.q * other.p
        s.q = self.q * other.q
        return s

r1 = Rational(2,3)
r2 = Rational(2,5)

sum = r1+r2
print(sum.p, sum.q)
```

we are writing a constructor and method through this way we add 2 rational numbers

```
16 15

Process finished with exit code 0
```

- You can also overload for **subtraction , multiplication , power , division** etc.

Binary Operators	
__add__()	Addition
__sub__()	Subtraction
__mul__()	Multiplication
__pow__()	Power
__truediv__()	Division
__floordiv__()	Floor Division
__mod__()	Remainder (modulo)
__lshift__()	Bitwise Left Shift
__rshift__()	Bitwise Right Shift
__and__()	Bitwise AND
__or__()	Bitwise OR
__xor__()	Bitwise XOR

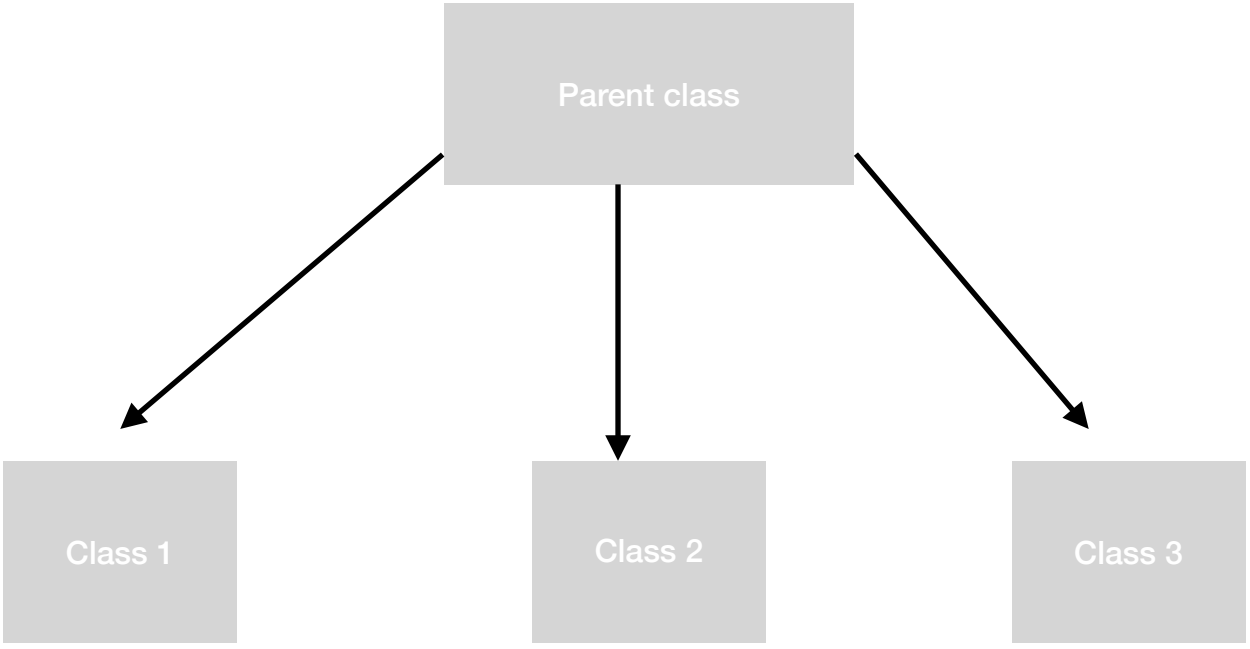
- Operator Overloading also works well for **comparison operators and assignment operators.**

Assignment Operators	
__iadd__()	equivalent to a += b
__isub__()	equivalent to a -= b
__imul__()	equivalent to a *= b
__idiv__()	equivalent to a /= b
__ifloordiv__()	equivalent to a //= b
__imod__()	equivalent to a %= b
__ipow__()	equivalent to a **= b
__ilshift__()	equivalent to a <<= b
__irshift__()	equivalent to a >>= b
__iand__()	equivalent to a &= b
__ior__()	equivalent to a |= b
__ixor__()	equivalent to a ^= b

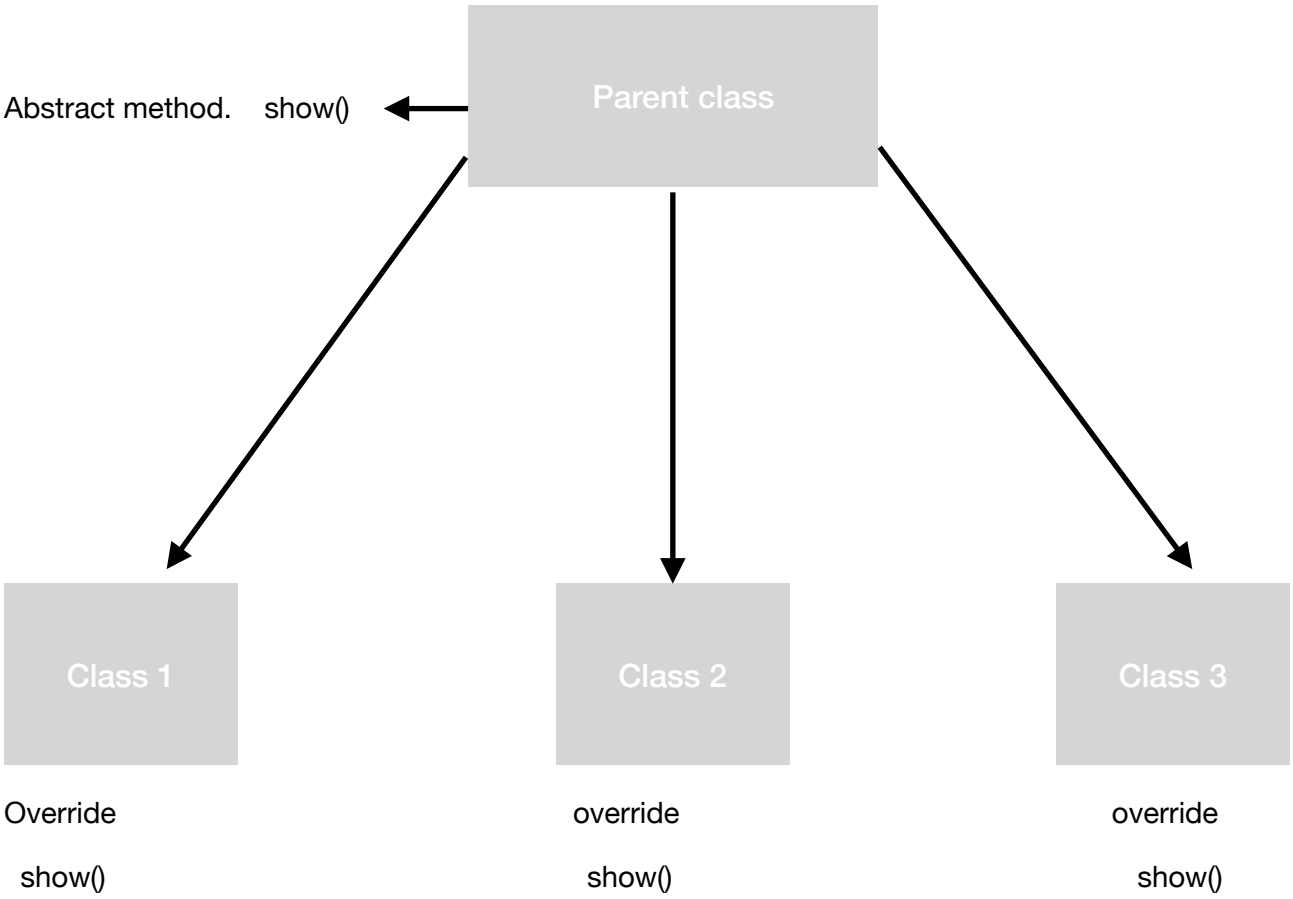
Comparison Operators	
__lt__()	Less than
__le__()	Less than or equal to
__eq__()	Equal to
__ne__()	Not equal to
__gt__()	Greater than
__ge__()	Greater than or equal to

Abstract classes and interface

- Instead of writing same method in each class and every class we can write a parent class and can inherit from parent class
- So, if we have any common code we can write it in a parent class.
- If we write a parent class then all these three class can inherit that from parent class
- As parent class is not directly used so, we can make it abstract
- Abstract means which cannot be instantiated.



- The whole and sole purpose of this class is to share the content with it's child classes this is one reason to write abstract classes
- Abstract classes cannot be instantiated but when we write the child classes for this abstract classes then we can create the instance of these child classes.
- But there is one more reason to write abstract classes.



- Classes must have some set of methods. One or more classes.
- Which is mandatory.
- We can force that classes to inherit them from parent class.

- If the body is not there then this is abstract method

- These classes must override the method in parent class so then that parent class is called as abstract class

Abstract: abstract class may have some abstract method and some concrete method

- Concrete method for sharing the code and abstract method for forcing the class to override method

Interface: interface means abstract class only but which is having only abstract class. Just for forcing child classes to override the method that is why we call it as an interface.

- Abstract classes interfaces are not directly supported by interpreter of python but it is provided as a module.

```python
from abc import ABC, abstractmethod

class Parent(ABC):#abstract class #any class inheriting abc becomes abstract class

    @abstractmethod
    def show(self):
        pass

    def display(self):#concrete method #a method which have body is called concrete
        print('Parent Display')


class Child(Parent):
    def show(self):#must override show as show method is declared as abstract in parent
        print('Show from Child')

c = Child
c.show()#calling method
c.display()
```

- Both are abstract it is nothing but interface. One as abstract and one as concrete
- Interface is also written in same way but all the methods are abstract.
- It is directly not available in python it is available through a module ABC

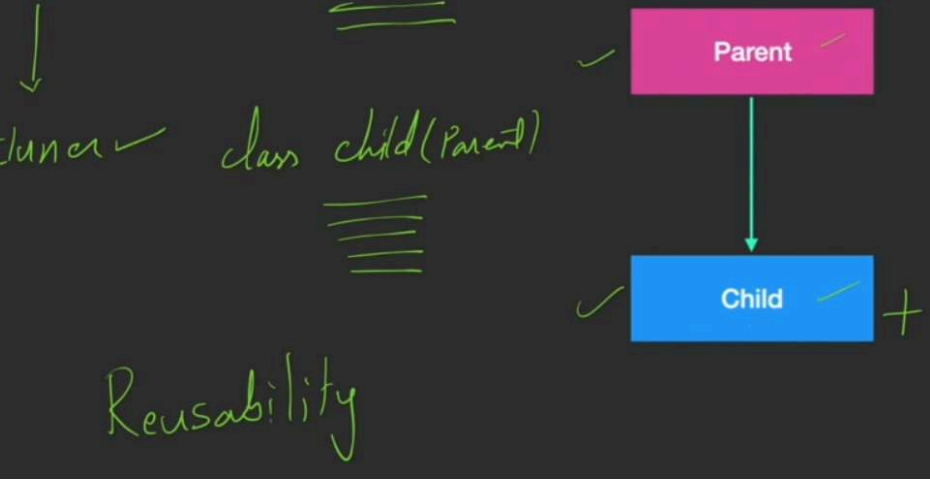
What is Inheritance?

innova

class Parent:

Fortuner

class child(Parent)



Reusability

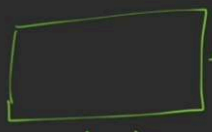
Python Course 2

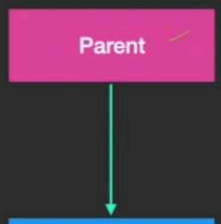
What is Inheritance?

object

```
class Rectangle:
    def __init__(self, l, b):
        self.length = l
        self.breadth = b
    def area(self):
        return self.length * self.breadth
    def perimeter(self):
        return 2 * (self.length + self.breadth)
```

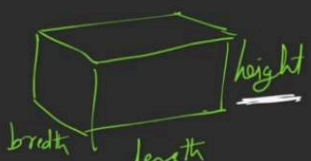
Rectangle






```
class cuboid(Rectangle):
    def __init__(self, h):
        self.height = h

    def volume(self):
        return self.length * self.breadth * self.height
```

Cuboid



Student Challenge

Inheritance

class Polygon

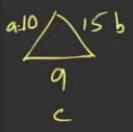
no_of_sides

sides = [5, 2, 5, 2]

→ __init__(self, ns, *sides)

class Triangle

area()



a = 10 b = 15 c = 9

$s = (a + b + c) / 2$

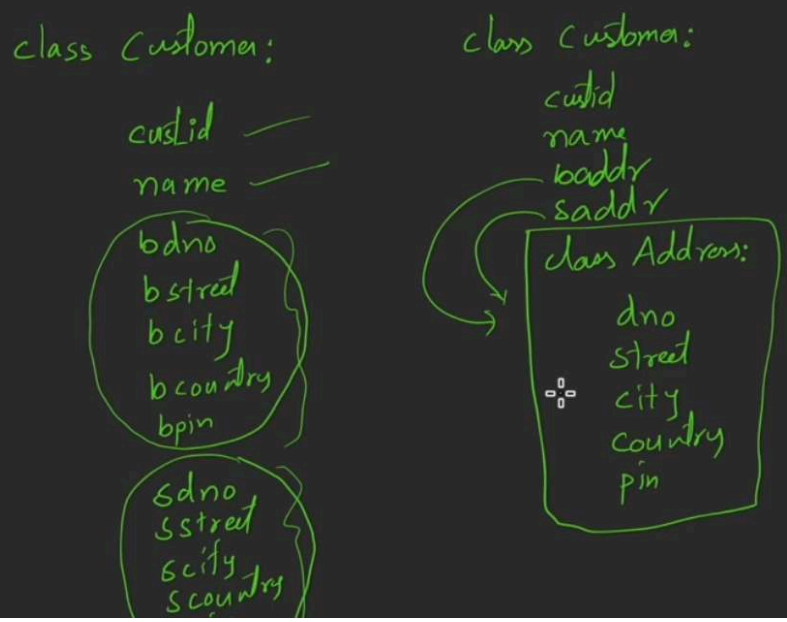
$area = \sqrt{s * (s-a) * (s-b) * (s-c)}$

__init__(self, ns, *sides)

Polygon.__init__(ns, *sides)

def area( )

Inner / Nested Classes



Student Challenge

Outer and Inner Class

class Course

- course_name
- course_duration
- books
- show_details()

class Book

- title
- __str__()
- str

```
__init__(self, nam, du, *book)
self.course_name
self.
self.books = [self.Book(ai) for ...]
```

Student Challenge

Inner Class

```
class Computer
    name
    CPU
    OS
    __str__()

class CPU
    make
    get_make()

class OS
    name
    get_name()
```

PCI

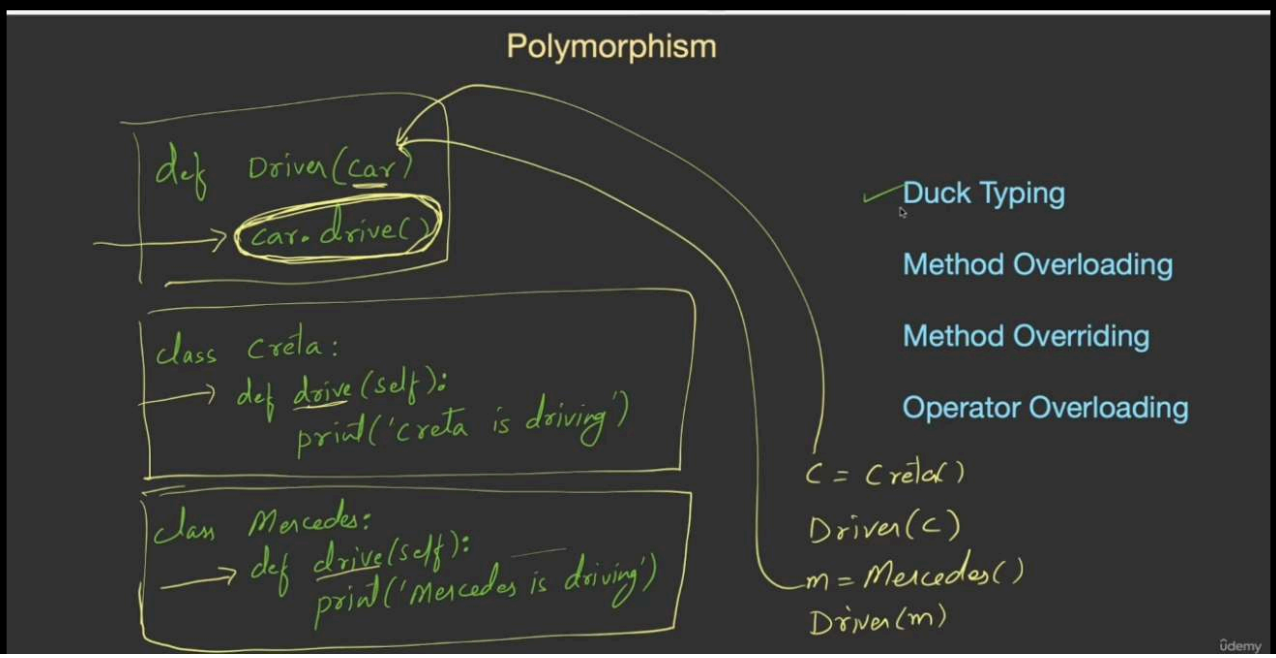
CPU
Intel
AMD
M1

OS
windows
Linux
macos

name, make, os
__init__(— — —)

name
cpu = CPU(—)
os = OS(—)

CPU
__init__()
make
get_make()


Polymorphism
def Driver(car)
car.drive()
class Creta:
def drive(self):
print('Creta is driving')
class Mercedes:
def drive(self):
print('Mercedes is driving')
C = Creta()
Driver(C)
m = Mercedes()
Driver(m)
Duck Typing
Method Overloading
Method Overriding
Operator Overloading
Udemy

Method Overloading

```
class Arith:
    def sum(x, y)
        return x+y
```

Hello world

```
a = Arith()
print (a.sum(10,5))
print (a.sum(8.5, 7.6))
print (a.sum('Hello', 'world'))
```

— 15

— 16.1

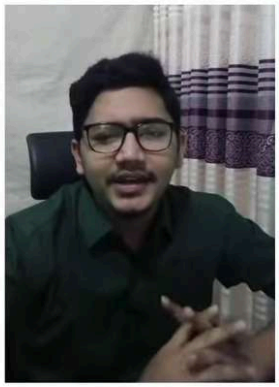
— HelloWorld

Operator

- We can use '+' operator to add numbers as well as to concatenate strings. Also it works for merging two lists.
- Reason behind this is the '+' operator is overloaded by ***int*** class and ***str*** class
- The same operator or function demonstrates different results for objects of different classes, this is called Operator Overloading

```
#adding integers
print(10 + 20)
Output:
30

#concatenate two strings
print('CSE' + '111')
Output:
CSE111
```

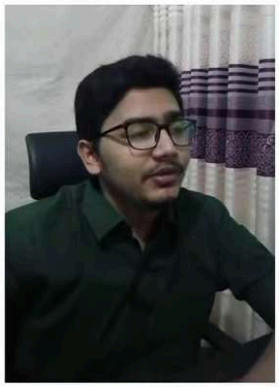
```
#Product of two numbers
print(10 * 20)
Output:
200

# Repeat the String
print('CSE' * 2)
Output:
CSECSE
```



Operator Overloading

```
class sum:
    def __init__(self, x, y):
        self.num = x + y

num1 = sum(10, 20)
num2 = sum(20, 30)
print(num1+num2)
```

```
Output:
TypeError: unsupported operand
type(s) for +: 'sum' and 'sum'
```



We are getting an error as Python don't know how to add two sum objects together.

This can be solved by using operator overloading.

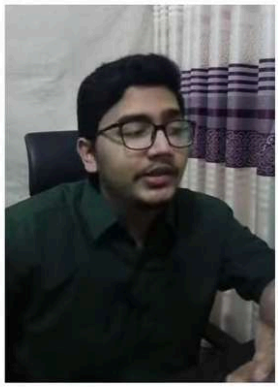
Operator Overloading (+)

- To overload the *(+)* operator, we need to implement __add__() method in the class

```
class data:
    def __init__(self, x):
        self.x = x

    # adding two objects
    def __add__(self, other):
        return self.x + other.x

num1 = data(10)
num2 = data(20)
str1 = data('cse')
str2 = data('111')

print(num1 + num2) #num1.__add__(num2)
print(str1 + str2) #str1.__add__(str2)
```

Output:

```
30
cse111
```



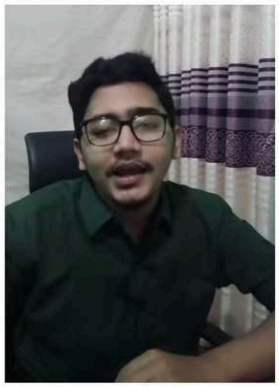
Operator Overloading (>)

- To overload the *(>)* operator, we need to implement __gt__() method in the class

```
class data:
    def __init__(self, x):
        self.x = x

    def __gt__(self, other):
        if(self.x > other.x):
            return True
        else:
            return False

num1 = data(10)
num2 = data(20)

if(num1>num2):  #num1.__gt__(num2)
    print('num1 is greater than num2')
else:
    print('num2 is greater than num1')
```

Output:

```
num2 is greater than num1
```



Operator Overloading (<)

- To overload the *(<)* operator, we need to implement __lt__() method in the class

```
class data:
    def __init__(self, x):
        self.x = x

    def __lt__(self, other):
        if(self.x < other.x):
            return "num1 is less than num2"
        else:
            return "num2 is less than num1"

num1 = data(10)
num2 = data(20)
print(num1 < num2) #num1.__lt__(num2)
```

```
Output:
num1 is less than num2
```

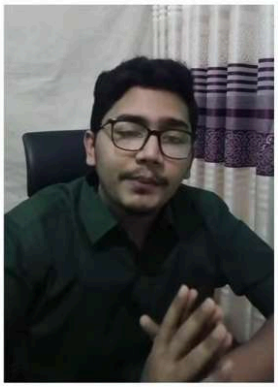
Operator Overloading (<)

- To overload the *(==)* operator, we need to implement __eq__() method in the class

```
class data:
    def __init__(self, x):
        self.x = x

    def __eq__(self, other):
        if(self.x == other.x):
            return "Both are equal"
        else:
            return "Not equal"

num1 = data(30)
num2 = data(30)
print(num1 == num2) #num1.__eq__(num2)
```

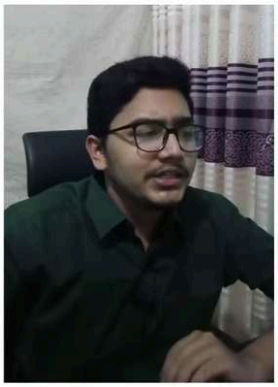
Output:

```
Both are equal
```



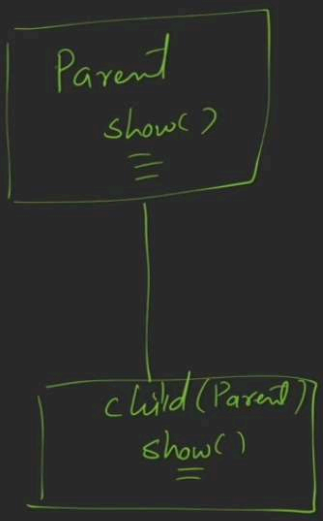
Some Python methods for operator overloading

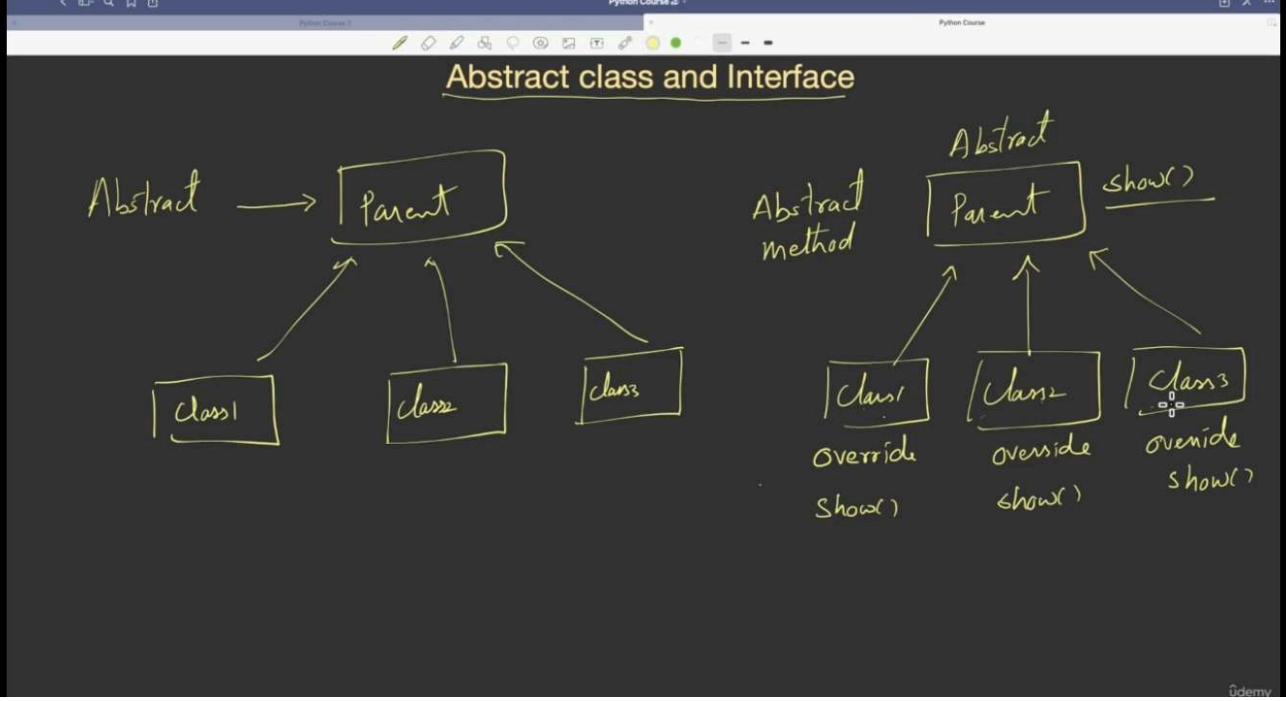
Operator	Expression	Internal Method
(+) Addition	n1 + n2	n1.__add__(n2)
(-) Subtraction	n1 - n2	n1.__sub__(n2)
(*) Multiplication	n1 * n2	n1.__mul__(n2)
(**) Power	n1 ** n2	n1.__pow__(n2)
(/) Division	n1 / n2	n1.__truediv__(n2)
(//) Floor Division	n1 // n2	n1.__floordiv__(n2)
(<) Less than	n1 < n2	n1.__lt__(n2)
(<=) Less than or equal to	n1 <= n2	n1.__le__(n2)
(==) Equal to	n1 == n2	n1.__eq__(n2)
(!=) Not equal to	n1 != n2	n1.__ne__(n2)
(>) Greater than	n1 > n2	n1.__gt__(n2)
(>=) Greater than or equal to	n1 >= n2	n1.__ge__(n2)



Method Overriding

```
class iPhone6:
    def home(self):
        print('home button is pressed')

class iPhoneX(iPhone6):
    def home(self):
        print('home is touched')
```




Abstract class and Interface
Abstract
Parent
class1
class2
class3
Abstract
Abstract method
Parent
show()
class1
class2
class3
Override show()
override show()
override show()
Udemy