why we use the object oriented programming ??

the problem with the procedural programming that we can't handle each of variables or entities which are initialised

into the programme

EX- you have to remember the student name variable... which would be difficult to remember that..

so ,that's why we use the oops.

object oriented programming --> we make the blueprint of any object or entity which is known as class

the real world entity which we use into that class that is object.

```
name = 'akanksha'
course = 'BCA'
location = 'jaipur'
roll no. = '34
#why we use the object oriented programming ??
# the problem with the procedural programming that we can't handle each of variables or entities whic
# into the programme
# EX- you have to remember the student name variable... which would be difficult to remember that..
# so ,that's why we use the oops.
```

```
name = 'akanksha'
course = 'BCA'
location = 'jaipur'
roll no. = '34
#why we use the object oriented programming ??
# the problem with the procedural programming that we can't handle each of variables or entities whic
# into the programme
# EX- you have to remember the student name variable... which would be difficult to remember that..
# so ,that's why we use the oops.
```

```
# here we have created class which has attributes name
class student:

  # BY DEFAULT akanksha
  # class attribute :- this class attributes is same for all the objects or instance
  name = 'akanksha'
  #constructor is automatically called when we create an object..
  # by giving one property or argument
  def __init__(self,location):
    self.attr_location = location
# here we have created a student object
obj1 = student('jaipur')
```

```
# here we have accessed the object's attribute
print(obj1.attr_location)
obj2 = student('delhi')
print(obj2.attr_location)
```

```
jaipur
delhi
```

```
# make a class of teacher ,experience ,name,domain
class teacher:
  def __init__(self,name,experience,domain):
    self.name = name
    self.experience = experience
    self.domain = domain
obj1 = teacher('akanksha',0,'Ai')
print(obj1.name)
print(obj1.experience)
print(obj1.domain)
```

```
akanksha
0
Ai
```

```
object,classes,self keyword,constructor
```

```
# classes :- blueprint of the object
class house:
  colour = 'blue'
  room = 5
  def display(self):
    print(self.colour)
    print(self.room)
house1 = house()
house2 = house()
house1.display()

# no_room = house.room
# print(no_room)
# print(colour)
```

```
blue
5
```

```
# object ke current reference ko btata hai
# self keyword is used to show the current object or instance location
class house:
  colour = 'blue'
  room = 5
  # constructor is automatically called when the object is crete
  def __init__(self):
    self.house_colour = user_colour
    print(self)
    house1 = house(green)
    house2 = house(red)
  print(house1.house_colour)
  print(house2.house_colour)
  print(house1.room)
  print(house2.room)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipython-input-3353548336.py in <cell line: 0>()
      1 # object ke current reference ko btata hai
      2 # self keyword is used to show the current object or instance location
----> 3 class house:
      4   colour = 'blue'
      5   room = 5

/tmp/ipython-input-3353548336.py in house()
     10     house1 = house(green)
     11     house2 = house(red)
---> 12   print(house1.house_colour)
     13   print(house2.house_colour)
     14   print(house1.room)
```

```
class student:
  college = 'IIT'
  def __init__(self,name,location):
    #instance attribute which is defined into the defined into the constructor method
    self.student_name = name
    self.student_location = location
  def display(self):
    print(f'student_name:{self.student_name}')
    print(f'student_location:{self.student_location}')
    print(f'college:{self.college}')
s1 = student('akanksha','jaipur')
print(s1.student_name)
```

```
akanksha
```

```
# create a class of employee
#class attribute company
#employee detail :- name,location,department,salary
# make a method to display all the details of the employee
class employee:
  comapany = 'regex software'
  def __init__(self,name,location,department,salary):
    self.name = name
    self.location = location
    self.department = department
    self.salary = salary
  def display(self):
    print(f'name:{self.name}')
    print(f'location:{self.location}')
    print(f'department:{self.department}')
    print(f'salary:{self.salary}')

emp1 = employee('Akanksha', 'Jaipur', 'Data Science', 30000)
emp1.display()
```

```
name:Akanksha
location:Jaipur
department:Data Science
salary:30000
```

```
#create a class of book with instance attribute
# title, writer ,published_on display the details with the display method
class book:
```

```
  def __init__(self,title,writer,published_on):
    self.title = title
    self.writer = writer
    self.published_on = published_on
  def display(self):
    print(f'title:{self.title}')
    print(f'writer:{self.writer}')
    print(f'published_on:{self.published_on}')
book1 = book('akanksha gupta', 'soft way', '2026')
book1.display()
```

```
title:akanksha gupta
writer:soft way
published_on:2026
```

```
# the main pillers of the oop is
# 1.inheritance
# 2.encapsulation
# 3.polymorphism
# 4.abstraction
```

```
# inheritance :- it is a one of its pillar which inherit the property from the parent class to the ch
# class this is parent class

class animal:
  def sound(self):
    print('animal makes sound!')
    #this is inherited class which is inherited by animal class

class dog(animal):
  pass
# animal().sound()

dog = dog()
dog.sound()
```

```
animal makes sound!
```

```
# there are two types inheritance is there
# 1.multiple inheritance
# 2.multilevel inheritance
```