oops_by_codeyug_2.py

```

# Topic: Polymorphism (poly - many, morphism - form) :- something having no of forms.

# Real life analogy:
#                          --You--
# In front of parent you talk ---------> study, career, exams etc
# In front of friends you talk --------> movies, Netflix, series,gf-bf etc
# :- according to sitution we adopt behavious/form --- one way to imagine polymorphism.

# example:
# + :- python object
print (10+20) #30
print("hello"+"welcome") #hellowelcome

# another example:
class Veh():
    def __init__(self,name,color,price):
        self.name = name
        self.color = color
        self.price = price

    def get_details (self):
        print("name is: ", self.name)
        print("color is:", self.color)
        print("price is:", self.price)

    def max_speed (self):
        print("maximum speed limit is 100")

    def gear(self):
        print("gear change is 6")

class Car(Veh):
    def max_speed(self):
        print("maximum speed limit is 140")

    def gear(self):
        print("gear change is 7")

V1=Veh("Truck", 'red', 200000000)
C1=Car("Car", 'white', 7000000)

V1.max_speed()       #---->> here max_speed() is the same method but a/c to object it will act diffrently.
C1.max_speed()

#_____________________________________________________________________________________________#


# Topics :  Over-riding Built in Functions using Magic Methods functionality
# - we can change the functionlity of built in function as per our need.
# - by using magic method convention we can override the built in functionality.
```

```
52
53  # - Example:
54
55  class Cart:
56      def _init__(self,basket1,basket2, basket3):
57          self.clothes-basket1
58          self.electronics-basket2
59          self.other-basket3
60      def __len__(self):
61          print("total numbe rof items in cart:")
62          return len(self.clothes)+len(self.electronics)+len(self.other)
63
64  shantanu = Cart(['pant', 'shirt', 't-shirt'], ['earphone', 'mobile'], ['chair'])
65  print (len (shantanu)) #6 -- this functionality would have not run if it has not been
    defined(over-riding) in class.
66
67
68
69  #_____________________________________________________________________________
    ____________________________________________________________-#
70
71
72  # Topic: Polymorphism in Functions and Objects
73
74  class BMW:
75      def fuel_type(self):
76          print("disel")
77
78      def max_speed(self):
79          print("max speed is 200")
80
81  class Ferrari:
82      def fuel_type(self):
83          print("petrol")
84
85      def max_speed(self):
86          print("max speed is 300")
87
88  def get_details(obj):
89      obj.fuel_type()
90      obj.max_speed()
91
92  bmw = BMW()
93  ferrari = Ferrari()
94
95  get_details(ferrari)
96
97
98  #_____________________________________________________________________________
    ____________________________________________________________-#
99
100
101 # Topics: Nested class
102 #  - The class which is declared inside another class.
103
104 #      class University:
105 #         #University class members
```

```python
#         class College:
#             #College class members

# - why it is required..?
#   When one class object cannot exist without another class object
#   i.e: Class Clock & class Cell, both have dependency of each other so here outter class is clock & inner class is cell

# Example1 :
class Outer:
    def __init__(self):
        print('outer class constructor executated!')
    def display(self):
        print('This is display method.')

    class Inner:
        def __init__(self):
            print('inner class constructor executated!')
        def show(self):
            print('This is show method.')

out = Outer()
inn = out.Inner()
out.display()

# Example2:

class Student:
    def __init__(self,name,roll,dd,mm,yy):
        self.name = name
        self.roll = roll
        self.dob = self.DOB(dd,mm,yy)           #--->> outside the class this is how we write: dob = std1.DOB()
                                                 #--->> self.dob -> saving into variable, self.DOB(dd,mm,yy)-->creating obj automatically
                                                                  #  whenever outer class created.
    def display(self):
        print(f"name of student is {self.name} and roll is {self.roll}")
        self.dob.display()

    class DOB:
        def __init__(self,dd,mm,yy):
            self.dd = dd
            self.mm = mm
            self.yy = yy

        def display(self):
            print(f"DOB is: {self.dd}/{self.mm}/{self.yy}")

std1 = Student('Ajay',101,18,2,2023)
std1.display()
```