oops_by_codeyug_1.py

```python
# accessing the attributes and methods of the class

class Employee:

    def __init__(self,salary,age):
        self.salary = salary
        self.age = age

    def display(self):
        return "The salary of the employee is {} and his age is {}".format(self.salary,self.age)


e1 = Employee(45000,21)
e2 = Employee(37000,23)

print(e2.display())

#_____________________________________________________________________________#

# Built in Class Functions

class Friends:

    def __init__(self,name,age):
        self.name = name
        self.age = age

f1 = Friends("harshad",23)
f2 = Friends("uddhal",53)

# print(getattr(f1,'name'))   #--> get attribute fetch the sp. attribute of mention object
# print(f2.name)
setattr(f2,'name','maitheli') #--> assign the value to atribute of object
# print(f2.name)
# print(f2.__dict__)
delattr(f2,'age')             #--> delete the mention attribute of mention object
# print(f2.__dict__)
print(hasattr(f2,'name'))     #--> return true/false if attribute for mention class is presentt or not.


#_____________________________________________________________________________#

#  Built in Class Attributes

class Friends:
    '''This class holds the information regading the name and age of the friends'''

    def __init__(self,name,age):
        self.name = name
```

```
50          self.age = age
51
52  f1 = Friends("harshad",23)
53  f2 = Friends("uddhal",53)
54
55  print(Friends.__dict__)      #--> displays the content of class
56  print(Friends.__doc__)       #--> display the comment line having purpose of the class
57  print(Friends.__name__)      #--> display the name of the class
58  print(Friends.__module__)    #--> diplay module name (file name)
59
60  #_____________________________________________________________________________
    ____________________________________________________________________________#
61
62  '''Inheritance Notes:
63     - child class can inherit the attributes and method from the parent class but parent class
    cant from child class.
64     - in case of the constructor, if child class dose not have the __init__ magic method it
    will look into parent class but if it dose have then
65       child class will prefer it own contructor before parent class. this is nown as
    constructor overriding.
66     - super() function, we can access the properties of parent class. if child and parent
    class both have constructor and child class instance also
67       wants to inherit the constructor from parent class then this function is used.'''
68
69  # ----> Super function()
70
71  class Computer:
72      def __init__(self):
73          self.ram = '8gb'
74          self.storage = '500gb'
75          print('class computer has executed.')
76
77  class Mobile(Computer):
78      def __init__(self):
79          super().__init__()
80          self.model = 'iphone X'
81          print('class Mobile has executed.')
82
83  apple = Mobile()
84  print(apple.__dict__)
85
86  # --->super fucntion with parametric constructor
87
88  class Computer:
89      def __init__(self,ram,storage):
90          self.ram = ram
91          self.storage = storage
92          print('class computer has executed.')
93
94  class Mobile(Computer):
95      def __init__(self,ram,storage):
96          super().__init__(ram,storage)
97          self.model = 'iphone X'
98          print('class Mobile has executed.')
99
100 apple = Mobile('8gb','512gb')
101 print(apple.__dict__)
```

```
102
103 '''with help of super function you can call the any property of parent class,method or
    attributes.'''
104
105 #_____________________________________________________________________________
    _____________________________________________________________________#
106
107 # Types of inheritance:
108 # 1) single inheritance - one parent and one child
109 # 2) multilevel inheritance - parent and child class further inherited into new class forming
    multiple level.
110
111 # below is the example of the multi level inheritance
112 class Parent:
113     name = 'Parent'
114
115 class Child_1(Parent):
116     middle_name = 'child_1'
117
118 class Child_2(Child_1):
119     surename = 'Child_2'
120
121 print(Child_2.surename)
122
123 # 3) hierarchical inheritance: single parent and multiple child
124 '''                 ______child_1
125     parent ------|
126                 ______child_2
127
128     child classes can access parent class but not a vice versa. and also child_1 nd child_2
    can not access each other.
129 '''
130
131 class Person:
132     def __init__(self,name,age):
133         self.name = name
134         self.age = age
135
136 class Employee(Person):
137     def __init__(self, name, age,salary):
138         super().__init__(name, age)
139         self.salary = salary
140
141 class Student(Person):
142     def __init__(self, name, age,marks):
143         super().__init__(name, age)
144         self.marks = marks
145
146 p1 = Person('Jay',23)
147 e1 = Employee('harshad',23,45000)
148 s1 = Student('akkshay',24,67)
149
150 # print(e1.marks) ----->> attributeError
151 print(s1.age)
152
153
154 # 4) multiple inheritance:
```

```
# - class derived from multiple derived classes
# - child class can access the both parent class property but parent classess can not of child or of each other.

'''  Parent_1_______
                    |_____ Child
     Parent_2_______|
'''
# --simple example:
class Country:
    office = 'india'

class State:
    # office = 'mumbai'
    pass

class District(State,Country):
    pass

d=District()
print(d.office)

# -- example with constructor:
class Country:
    def __init__(self):
        print('Country class constructor')
        self.office = 'India'

class State:
    def __init__(self):
        super().__init__()
        print('State class constructor')
        self.office = 'Mumbai'

class District(State,Country):    # --> parent classes can be multiple. access flow from left to right.
    def __init__(self):
        super().__init__()
        print('District class constructor')
        self.office = 'Pune'

d=District()
print(d.__dict__)

# 5) Hybrid inheritance
''' Parent_1_______                    _______child_1
                    |_____ Child____|
    Parent_2_______|               |_______child_2

    - mixture of multiple inheritance and hierachical inheritance
'''

#_____________________________________________________________________________________________#

'''
Topic: Encapsulation
```

```
Access Modifiers in Python :
    - Generally, we restrict data access outside the class in encapsulation.
    - Encapsulation can be achieved by declaring the data members and methods of a class as private.
    - Three access specifiers:- public, private, protected(not usually in use.)

    - Public member:- Accessible anywhere by using object reference.
    - Private member:- Accessible within the class. Accessible via methods only.(can make attribute private by settting
      2 underscore just before the attribute. i.e: self.__revenue = 1,00,000). same is for methods but not in much use.
    - Protected member:- Accessible within class and it's subclasses(can make attribute protected by settting
      1 underscore just before the attribute. i.e: self._revenue = 1,00,000)
'''

class Finance:
    def __init__(self):
        self.revenue = 100000
        self.no_of_employee = 24

f1 = Finance()

class Hr:
    def __init__(self):
        self.salary = 5000
        print(f"before changing the value {f1.revenue}")
        f1.revenue = 20000
        print(f"after changing the value {f1.revenue}")

h1 = Hr()

'''- need of encapsulation: in above example we can access finance class attibute by the object in the scope
   of another Hr class. sometime this access can be harmfully or we want make that attribute/property to access
   within the particular class only, so with the help of encapsulation we make properties of class accessible to
   particular scope. they can only be access by class methods only.

   - in python there's no pure encapsulation, it is just to restrict accessing directly. ofcourse there r solution and modules for
    pure encapsulation in pythin. we can access the property outsite the class by some logic.
    encapsulated private property stored in this way --> i.e.: '_Finance__revenue': 100000, -> _classname__variablename (name mangling).
    with help of this we can access the property outside the class.

      '''

class Finance:
    def __init__(self):
        self.__revenue = 100000
        self.__no_of_employee = 24

    def display(self):
        print(self.__revenue)

f1 = Finance()
```

```
258
259 class Hr:
260     def __init__(self):
261         self.salary = 5000
262         # print(f"before changing the value {f1.__revenue}")
263         # f1.__revenue = 20000                                    #--->> if we try to
    access or change the value of property will throw an error!
264         # print(f"after changing the value {f1.__revenue}")
265
266 h1 = Hr()
267 f1.display()   # it can only be access by the particular class method.
268 print(f1.__dict__)
269
270 # - accessing the attibute outside the class with logic:
271
272 class Finance:
273     def __init__(self):
274         self.__revenue = 100000
275         self.__no_of_employee = 24
276
277     def display(self):
278         print(self.__revenue)
279
280 f1 = Finance()
281
282 class Hr:
283     def __init__(self):
284         self.salary = 5000
285         print(f"before changing the value {f1._Finance__revenue}")
286         f1._Finance__revenue = 20000                                  #--->> if we try to
    access or change the value of property will throw an error!
287         print(f"after changing the value {f1._Finance__revenue}")
288
289 h1 = Hr()
290 f1.display()           # it can only be access by the particular class method.
291 print(f1.__dict__)
292
293 #_____________________________________________________________________________
    _____________________________________________________________________________#
```