LAB ACTIVITY 9

NAME : SONALI R; REG NO : 22MID0061; COURSE CODE: CSI3007; LAB : L27+L28;

DATE : 12/08/2025

Class variables

In [28]:
```
class Emp1:
    no_of_employees = 0
    raise_amount = 1.04

    def __init__(self, first, last, pay):
        self.first = first
        self.last = last
        self.pay = pay

    no_of_employees += 1

    def apply_raise(self):
        self.pay = int(self.pay * Emp1.raise_amount)
```

In [29]:
```
# class variable before creating instances
print(Emp1.no_of_employees)
```

```
1
```

In [30]:
```
# Creating the Instances
e1 = Emp1("Sonali","Rajaram","500000")
e2 = Emp1("Kamesh","Rajaram","600000")
```

In [31]:
```
# class variable after creating instances
print(Emp1.no_of_employees)
```
```
1
```

In [32]:
```
# namespace for class
print(Emp1.__dict__)
```
```
{'__module__': '__main__', 'no_of_employees': 1, 'raise_amount': 1.04, '__init__': <function Emp1.__init__ at 0x75d5
233e4220>, 'apply_raise': <function Emp1.apply_raise at 0x75d5233e5120>, '__dict__': <attribute '__dict__' of 'Emp1'
objects>, '__weakref__': <attribute '__weakref__' of 'Emp1' objects>, '__doc__': None}
```

In [33]:
```
# namespace of the first instance e1 before it accessing the class variable
print(e1.__dict__)
```
```
{'first': 'Sonali', 'last': 'Rajaram', 'pay': '500000'}
```

In [34]:
```
# First Instance accessing and changing the class variable
e1.raise_amount = 1.05
```

In [35]:
```
# Namespace of the first instance e1 after it accessing the class variable
print(e1.__dict__)
```
```
{'first': 'Sonali', 'last': 'Rajaram', 'pay': '500000', 'raise_amount': 1.05}
```

In [36]:
```
# Printing the class variable after the first instance changing it
print(Emp1.raise_amount)
```
```
1.04
```

In [37]:
```
# printing the class variable using first instance
print(e1.raise_amount)
```
```
1.05
```

In [38]:
```
# printing the class variable using second instance
print(e2.raise_amount)
```
```
1.04
```

In [64]:
```
# Namespace of the second instance e2
print(e2.__dict__)
```

```
{'first': 'Kamesh', 'last': 'Rajaram', 'pay': '600000'}
```

Task 2: Create the class and count the number of instances created

In [66]:
```
class Employee:
    # Class variable to track the total number of employees
    num_of_employees = 0

    def __init__(self, first_name, last_name, pay):
        self.first_name = first_name
        self.last_name = last_name
        self.pay = pay
        self.email = f"{first_name}.{last_name}@company.com"

        # Increment the employee count every time a new employee is created
        Employee.num_of_employees += 1

    def fullname(self):
        return f"{self.first_name} {self.last_name}"
```

In [67]:
```
# object creation and accessing the members and member functions of the class
print(f"Initial employee count: {Employee.num_of_employees}")  # Accessing the class variable
```

```
Initial employee count: 0
```

In [68]:
```
# object creation
emp_1 = Employee('Elon', 'Musk', 50000)
emp_2 = Employee('Donald', 'Trump', 60000)
```

In [69]:
```
# Accessing the class variable after object creation
print(f"Current employee count: {Employee.num_of_employees}")
```

```
Current employee count: 2
```

Basic Inheritance

In [51]:
```
class Parent:        # Parent class
    def greet(self):
        print("Hello from parent")

class Child(Parent):   # Child class inheriting from the parents
    pass
```

In [52]:
```
obj = Child()  # creating the object for the child class
obj.greet()  # child class accessing the parent class method
```

```
Hello from parent
```

Using super() to call a method from parent class

In [55]:
```
class Parent:        # Parent class
    def greet(self):
        print("Hello from parent")

class Child(Parent):   # Child class inheriting from the parents
    def greet(self):
        super().greet() # calls Parent's greet
        print("Hello from Child")
```

In [56]:
```
obj = Child()  # creating the object for the child class
obj.greet()  # child class accessing method
```

```
Hello from parent
Hello from Child
```

Task 1: Multiple Inheritance and multiple child classes accessing the class variable

In [92]:
```
class Countable:
    """
    A mixin class to add object counting functionality.
    """
    object_count = 0  # Class variable to store the count

    def __init__(self, *args, **kwargs):
        # Increment the count when a new object is created
        Countable.object_count += 1
        super().__init__(*args, **kwargs)
```

In [93]:
```
class Person:
    def __init__(self, name):
        self.name = name
```

In [94]:
```
class Employee(Countable, Person): # Multiple inheritance
    def __init__(self, name, employee_id):
        # Call parent constructors using super()
        super().__init__(name=name)
        self.employee_id = employee_id
```

In [95]:
```
# Accessing the class variable before creating the objects
print(f"Number of Employee objects created: {Employee.object_count}")
```

```
Number of Employee objects created: 0
```

In [96]:
```
# Creating Employee objects
employee1 = Employee("Alice", "E001")
employee2 = Employee("Bob", "E002")
employee3 = Employee("Charlie", "E003")
```

In [97]:
```
# Accessing the class variable to get the count
print(f"Number of Employee objects created: {Employee.object_count}")
```

```
Number of Employee objects created: 3
```

In [98]:
```
# We can also access it through an instance, but it's recommended to use the class name
print(f"Number of Employee objects created (via instance): {employee1.object_count}")
```

```
Number of Employee objects created (via instance): 3
```

Task 3: Create the class that is inheriting from the class Emp1

In [58]:
```
## Task 3
# child class accessing the Emp1 class
class Emp2(Emp1):
    def __init__(self, first, last, pay):
        # Call parent constructors using super()
        super().__init__(first, last, pay)
    def apply_raise(self):
        super().apply_raise()
```

In [59]:
```
e3 = Emp2("Sonali","R",100000000) # creating the instance for child class inheriting from Emp1 class
```

In [60]:
```
e3.apply_raise()   # Instance accessing the inherited method
```

In [61]:
```
print(e3.pay)    # Object accessing the instance variable
```

```
104000000
```

In [62]:
```
print(e3.raise_amount)  # Object accessing the class variable
```

```
1.04
```

Multiple Inheritance

In [86]:
```
class A:
    def method_a(self):
        print("Method A")

class B:
    def method_b(self):
        print("Method B")

class C(A,B):  # Class inheriting from A and B class
    pass
```

In [87]:
```
obj = C()
obj.method_a()  # child class accessing parent class methods
obj.method_b()
```

```
Method A
Method B
```

Multi Level Inheritance

In [88]:
```
class A:
    def method_a(self):
        print("Method A")

class B(A):                 # Class inheriting from A
    def method_b(self):
        super().method_a()
        print("Method B")

class C(B):               # Class inheriting from B
    def method_c(self):
        super().method_b()
        print("Method C")
```

In [89]:
```
obj = C()
obj.method_c()  # object accessing its method
```

```
Method A
Method B
Method C
```

Understanding that Inheritance is Uni-Directional

In [80]:
```
class Contact:
    all_contacts = []

    def __init__(self, name, email):
        self.name = name
        self.email = email
        Contact.all_contacts.append(self)
```

In [81]:
```
class Supplier(Contact):
    def order(self,order):
        print("Send""{} order to {}".format(self,self.name))
```

In [82]:
```
c = Contact("Sonali","sonali@gmail.com")
s = Supplier("Kamesh","kamesh@gmail.com")
print(c.name,c.email,s.name, s.email)
```

```
Sonali sonali@gmail.com Kamesh kamesh@gmail.com
```

In [83]:
```
c.all_contacts
```

Out[83]:
```
[<__main__.Contact at 0x75d52327e4b0>, <__main__.Supplier at 0x75d52327c140>]
```

In [90]:
```
c.order("I need pliers")   # parent class has no access to child class members and member functions (It is Uni-dire
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[90], line 1
----> 1 c.order("I need pliers")

AttributeError: 'Contact' object has no attribute 'order'
```

In [91]:
```
s.order("I need pliers")  # fixing the above error
```

```
Send<__main__.Supplier object at 0x75d52327c140> order to Kamesh
```

Extending the built-in class

In [99]:
```
class ContactList(list):
    def search(self,name):
        """Return all contacts that contain the search value in their name"""
        matching_contacts = []
        for contact in self:
            if name in contact.name:
                matching_contacts.append(contact)
        return matching_contacts
```

In [101…
```
class Contact:
    all_contacts = ContactList()    # shared list of all contacts

    def __init__(self, name, email):
        self.name = name
        self.email = email
        Contact.all_contacts.append(self)  # Add self to the shared contact list
```

In [102…

```
c1 = Contact("John A","johna@example.net")
c2 = Contact("John B","johnb@example.net")
c3 = Contact("Jenna C","jennac@example.net")
[c.name for c in Contact.all_contacts.search('John')]
```

Out[102…

```
['John A', 'John B']
```

In [ ]: