UNIVERSITY OF ENGINEERING AND TECHNOLOGY PESHAWAR, JALOZAI CAMPUS

Lab 7: Inheritance

Lab Title: EE-271 "OOP & Data Structures Lab"

Time: 10 min/ Task

Lab report task

1. Make different objects of different classes and call different method and observe the output.

```
class Employee:#STILL LECTURE 23 MULTI LEVEL INHERITANCE
    raise_amount=1.1
    total_emp=0
    def __init__(self,fname,lname,pay):
        self.fname=fname
        self.lname=lname
        self.pay=pay
        self.email=f"{self.fname.lower()}.{self.lname.lower()}@uet.edu.pk"
        Employee.total_emp+=1
    def raise_pay(self):
        self.pay*=self.raise_amount

    def __repr__(self):
        return f'{self.fname}{self.lname}'



class Instructor(Employee):
    def __init__(self,fname,lname,pay,design):
        super().__init__(fname,lname,pay)
        self.design=design
        self.courses=[]
    def assignCourse(self,*Subj):
        self.courses=list(set(self.courses+list(Subj)))


class LabDirector(Instructor):
    def __init__(self,fname,lname,pay,design,lab,*labEmp):
        super().__init__(fname,lname,pay,design)
        self.lab=lab
        self.labEmp=(labEmp)
    def assignCourse(self,*subj):
        super().assignCourse(*subj)
        for ins in self.labEmp:
            ins.assignCourse(*subj)

    def addlabEmployee(self,ins):
        self.labEmp.append(ins)
```

```
    def droplabEmployee(self,ins):
        self.labEmp.remove(ins)


class AdminStaff(Employee):#Inherit From Parent Class'''
    raise_amount=1.5
    allAdminStaff=[]
    def __init__(self,fname,lname,pay,team=None):
        super().__init__(fname,lname,pay)
        self.team=team
        self.task=[]
        AdminStaff.allAdminStaff.append(self)
    def assignTask(self,*Task):
        self.task=list(set(self.task+list(Task)))

class AdminOfficer(AdminStaff):
    def __init__(self,fname,lname,pay,team):
        super().__init__(fname,lname,pay,team)
    @property
    def teamMem(self):
        return list(filter(lambda s:s.team==self.team,AdminStaff.allAdminStaff))
```

Lab work task

1. Inheritance is the process by which one class takes on the attributes and methods of another. Newly formed classes are called **child classes**, and the classes that you derive child classes from are called **parent classes**.
    - You inherit from a parent class by including the name of the parent class inside parentheses after the class name but before the colon terminating the class definition.

```
# inheritance.py

class Parent:
    hair_color = "brown"


class Child(Parent):
    pass
```

a. Run the code.

```
az = Parent()
az.hair_color
```

b. Run the code and comment on what you observed.

```
az = Child()
az.hair_color
```

2. **Overridden** the hair color attribute that you inherited from your parents:

```
class Parent:
    hair_color = "brown"

class Child(Parent):
    hair_color = "purple"
```

a. Run the code.

```
az = Parent()
az.hair_color
```

b. Run the code and comment on what you observed. How this result is different from the task number 1 and why?

```
az = Child()
az.hair_color
```

3. **Extended your attributes because you've added an attribute that your parents don't have:**

```
class Parent:
    speaks = ["English"]

class Child(Parent):
    def __init__(self):
        super().__init__()
        self.speaks.append("German")
```

a. Run the code.

```
az = Child()
az.speaks
```

4. The dog class is given below.

i. A simple dog class is given below.

```
# dog.py

class Dog:
    species = "Canis familiaris"

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

    def __str__(self):
        return f"{self.name} is {self.age} years old"

    def speak(self, sound):
        return f"The parrent class method call..... {self.name} says {sound}"
```

a. Make the following object of the dog class.

```
miles = Dog("Miles", 4)
buddy = Dog("Buddy", 9)
jack = Dog("Jack", 3)
jim = Dog("Jim", 5)
```

b. Run the following code.

```
print(miles)
jack.speak()
jim.speak('yap yap')
```

ii. Modify the Dog class in the editor by adding a .breed attribute.

```
# dog.py

class Dog:
    species = "Canis familiaris"

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

    def __str__(self):
        return f"{self.name} is {self.age} years old"

    def speak(self, sound):
        return f"{self.name} says {sound}"
```

a. Make the following dog instances.

```
miles = Dog("Miles", 4, "Jack Russell Terrier")
buddy = Dog("Buddy", 9, "Dachshund")
jack = Dog("Jack", 3, "Bulldog")
jim = Dog("Jim", 5, "Bulldog")
```

b. Using just the Dog class, you must supply a string for the sound argument of .speak() every time you call it on a Dog instance:

```
buddy.speak("Yap")
jim.speak("Woof")
jack.speak("Woof")
```

iii. Some Derived classes from the Dog class are given below in i.

```
class JackRussellTerrier(Dog):
    pass

class Dachshund(Dog):
```

```
    pass

class Bulldog(Dog):
    pass
```

a. The following is the object instantiation from the child class.

```
miles = JackRussellTerrier("Miles", 4)
buddy = Dachshund("Buddy", 9)
jack = Bulldog("Jack", 3)
jim = Bulldog("Jim", 5)
```

b. Run the following code and observe.

```
miles.species
buddy.name
print(jack)
jim.speak("Woof")
```

iv. One thing to keep in mind about class inheritance is that changes to the parent class automatically propagate to child classes. This occurs as long as the attribute or method being changed isn't overridden in the child class.
Following is the class derived from the base class Dog. The default parameter, change says to bark in the speak.

Note: More generally, all objects created from a child class are instances of the parent class.

```
class JackRussellTerrier(Dog):
    def speak(self, sound="Arf"):
        return f"Chiled class call ….{self.name} says {sound}"
```

a. Run the following code and observe.

```
miles = JackRussellTerrier("Miles", 4)
miles.speak()
```

b. Run the following.

```
miles = Dachshund("Miles", 4)
miles.speak('aaaaaaaaa')
```

c. Run the following.

```
miles = Bulldog("Miles", 4)
miles.speak('bbbbbbbbbbbbb')
```

d. Modify the other two child classes in part iii i.e. `Dachshund and Bulldog.` Update like in part I the `JackRussellTerrier` is modified.

e. You can access the parent class from inside a method of a child class by using `super()`:

```
class JackRussellTerrier(Dog):
```

```
    def speak(self, sound="Arf"):
        return super().speak(sound)
```

- Run the following.

```
miles = JackRussellTerrier("Miles", 4)
miles.speak()
```

Encouraging:

1. Person and Employ
   If your subclasses provide a .__init__() method, then this method must explicitly call the base class's .__init__() method with appropriate arguments to ensure the correct initialization of instances. To do this, you should use the built-in super() function.
   Run and understand the following code. Also, print the attributes.

```
class Person:
    def __init__(self, name, birth_date):
        self.name = name
        self.birth_date = birth_date
class Employee(Person):
    def __init__(self, name, birth_date, position):
        super().__init__(name, birth_date)
        self.position = position

john = Employee("John Doe", "2001-02-07", "Python Developer")
```

2. Consider the following class and sub-class.

```
class Contact:
    all_contacts = []
    def __init__(self, name, email):
        self.name = name
        self.email = email
        Contact.all_contacts.append(self)
class Supplier(Contact):
    def order(self, order):
        print("If this were a real system we would send '{}' order
to '{}'".format(order, self.name))
```

Instantiate two objects c and s.

```
c = Contact("Some Body", "somebody@example.net")

s = Supplier("Sup Plier", "supplier@example.net")
```

a. Print the attributes of c and s.
b. Call the order method on c.
c. Call the order method on s. Observe the difference.

Note:

- Supplier class can do everything a contact can do (including adding itself to the list of all_contacts) and all the special things it needs to handle as a supplier.

3. **Extending built-ins:** List is a built-in class. In the following code block "`ContactList` "is derived.

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

   a. Make the following three objects.
      c1 = Contact("John A", "johna@example.net")
      c2 = Contact("John B", "johnb@example.net")
      c3 = Contact("Jenna C", "jennac@example.net")
   b. Make a list of names that contain John. For this use the search method.

Note:

- In the `Contact` class seen earlier, we are adding contacts to a list of all contacts. What if we also wanted to search that list by name? Well, we could add a method on the `Contact` class to search it, but it feels like this method actually belongs to the list itself.
- Instead of instantiating a normal list as our class variable, we create a new `ContactList` class that extends the built-in `list`. Then, we instantiate this subclass as our `all_contacts` list.

4. **Overriding and super**

Consider the following `contact` class allows only a name and an e-mail address.

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

This may be sufficient for most contacts, but what if we want to add a phone number for our close friends?

To add a new attribute phone can be done easily by just setting a `phone` attribute on the contact after it is constructed.

```
class Friend(Contact):
    def __init__(self, name, email, phone):
        self.name = name
        self.email = email
```

```
                self.phone = phone
```

a. Make an object Friend class and print its attributes.

What we really need is a way to execute the original `__init__` method on the `Contact` class. This is what the `super` function does; it returns the object as an instance of the parent class, allowing us to call the parent method directly:

```
class Friend(Contact):
    def __init__(self, name, email, phone):
        super().__init__(name, email)
        self.phone = phone
```

This example first gets the instance of the parent object using `super`, and calls `__init__` on that object, passing in the expected arguments. It then does its own initialization, namely, setting the `phone` attribute.

b. Make an object Friend class and print its attributes.

Note:

- But if we want to make this third variable available on initialization, we have to override `__init__`. Overriding means altering or replacing a method of the superclass with a new method (with the same name) in the subclass. No special syntax is needed to do this; the subclass's newly created method is automatically called instead of the superclass's method.
- Any method can be overridden, not just `__init__`. Before we go on, however, we need to address some problems in this example. Our `Contact` and `Friend` classes have duplicate code to set up the `name` and `email` properties; this can make code maintenance complicated as we have to update the code in two or more places. More alarmingly, our `Friend` class is neglecting to add itself to the `all_contacts` list we have created on the `Contact` class.
- A `super()` call can be made inside any method, not just `__init__`. This means all methods can be modified via overriding and calls to `super`. The call to `super` can also be made at any point in the method; we don't have to make the call as the first line in the method. For example, we may need to manipulate or validate incoming parameters before forwarding them to the superclass.

Reading Section

a. https://realpython.com/python3-object-oriented-programming/#how-do-you-inherit-from-another-class-in-python
b. https://realpython.com/lessons/multiple-inheritance-python/
c. https://www.youtube.com/watch?v=8eh0ahzZ2nQ&list=PLWF9TXck7O_zuU2_BVUTrmGMCXYSYzjku&index=27
d. https://github.com/MAN1986/OOP-in-Python