|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C:\Users\Admin\Desktop\download.jpg | USMAN INSTITUTE OF TECHNOLOGY | | | | | |  |
|  |  |  |  |  |  |  |  |
|  | Department of Computer Science  CS121 Object Oriented Programming | | | | | |  |
|  |  | Lab # 11  Polymorphism | | | |  |  |
|  | Objective:  This experiment introduces the students to the concept of Polymorphism and its various application areas | | | | | |  |
|  | **Name of Student:**  **Roll No: Sec.**  **Date of Experiment:** | | | | | |  |
|  | **Marks Obtained/Remarks:**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Signature:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | |  |

**Lab 11: Polymorphism**

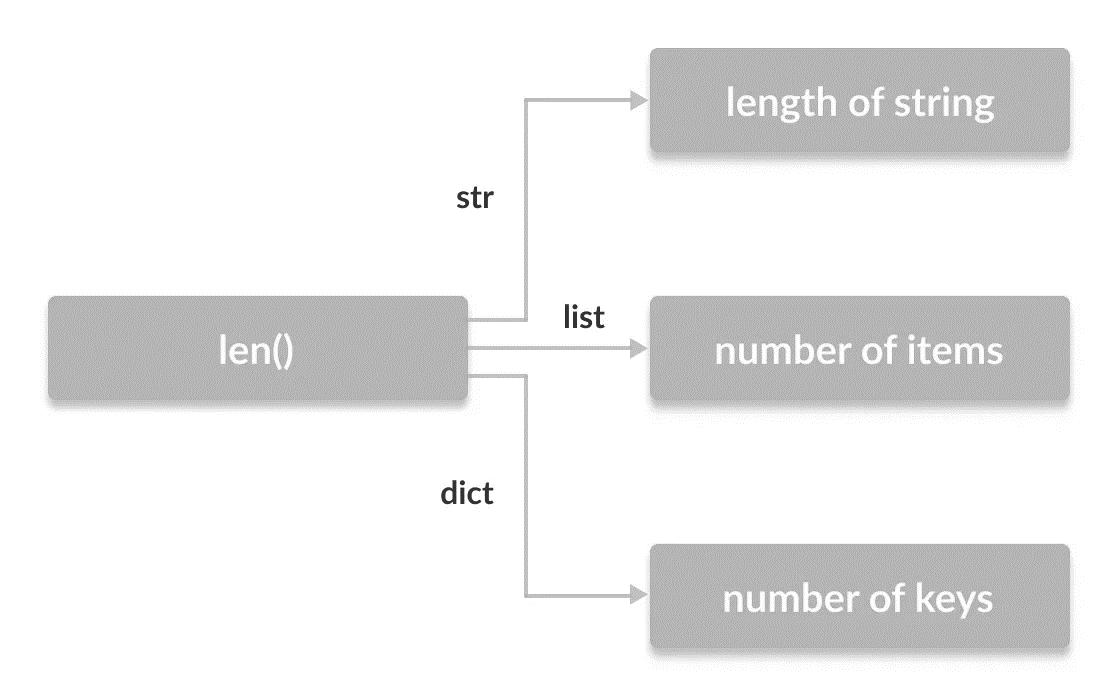
One of the pillar of Object-Oriented Programming. The literal meaning of Polymorphism is to have many forms or the condition of occurrence in different forms. Polymorphism is a very important concept in Object Oriented Programming and it refers to the use of a single type entity (operator, method or object) to represent different types in different scenarios.

**Basic Polymorphism – Operator Overloading**

A single operator can be used to carry out different operations for distinct data types. It is the simplest occurrence of polymorphism in Python. For example, (+) operator is extensively used in Python. However, it does not have a single use. On integer data type the operator performs an arithmetic addition whereas on String data type the same operator performs concatenation operation.

**Function Polymorphism**

Many data types in Python, like String, List, Tuple, Set, and Dictionary work with the len() function. However, the function returns specific information depending on the specific data types.



**Class Polymorphism**

The concept is used while creating class methods as Python allows different classes to have methods with the same name. We can then later generalize calling these methods by disregarding the object we are working with.

*Example 1: Two unrelated classes*

class Cat:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def info(self):

print(f"I am a cat. My name is {self.name}. I am {self.age} years old.")

def make\_sound(self):

print("Meow")

* class Dog:
* def \_\_init\_\_(self, name, age):
* self.name = name
* self.age = age
* def info(self):
* print(f"I am a dog. My name is {self.name}. I am {self.age} years old.")
* def make\_sound(self):
* print("Bark")
* cat1 = Cat("Kitty", 2.5)
* dog1 = Dog("Fluffy", 4)
* for animal in (cat1, dog1):
* animal.makeSound()
* animal.info()
* animal.makeSound()

In previous example two classes Cat and Dog are defined. They share a similar structure with the same method names info() and makeSound(). However, neither of the them have any common superclass nor the classes are linked together in any way. When objects of these classes are packed into a tuple then because of polymorphism it becomes possible to iterate through it using a common variable

**Polymorphism and Inheritance**

We know that when a class is extended then the child class inherit methods and attributes from the parent class. It is also possible that certain methods of the parent class may be redefined specifically to fit the child – concept known as the method overriding. In such a scenario, Polymorphism allow to access the overridden methods that have the same name as the parent class.

*Example 2:*

class Shape:

def \_\_init\_\_(self, name):

self.name = name

def area(self):

pass

def fact(self):

return "I am a two-dimensional shape."

def \_\_str\_\_(self):

return self.name

* class Square(Shape):
* def \_\_init\_\_(self, length):
* super().\_\_init\_\_("Square")
* self.length = length
* def area(self):
* return self.length\*\*2
* def fact(self):
* return "Squares have each angle equal to 90 degrees."
* class Circle(Shape):
* def \_\_init\_\_(self, radius):
* super().\_\_init\_\_("Circle")
* self.radius = radius
* def area(self):
* return pi\*self.radius\*\*2
* a = Square(4)
* b = Circle(7)
* print(b)
* print(b.fact())
* print(a.fact())
* print(b.area())

Output

Circle

I am a two-dimensional shape.

Squares have each angle equal to 90 degrees.

153.93804002589985

We can observe that \_\_str\_\_() method is not overridden in the child class and hence it is used from the parent class. Due to polymorphism, Python interpreter automatically recognizes that the fact() method for object a(Square class) is overridden. So, it uses the one defined in the child class. On the other hand, since the fact() method for object **b** (Circle class) isn't overridden, it is used from the Parent Shape class.

**isinstance() Function**

The isinstance() function determines whether an object is an instance of a class. The general notation is as follows:

isinstance(object, ClassName)

*Example 3:*

* class Tomato():
* def type(self):
* print("Vegetable")
* def color(self):
* print("Red")
* def taste(self):
* print(“Sweet")
* class Apple():
* def type(self):
* print("Fruit")
* def color(self):
* print("Red")
* def func(obj):
* obj.type()
* obj.color()
* if isinstance(obj, Tomato)
* obj.taste()
* obj\_tomato = Tomato()
* obj\_apple = Apple()
* func(obj\_tomato)
* func(obj\_apple)

# Student Exercise

Following UML diagram represent an Employee class and Salaried Employee and Hourly Employee classes being inherited from it

|  |
| --- |
| ***Employee*** |
| - name: String  - cnic: String |
| Employee(name: String, cnic: String)  *monthlyEarnings(): float*  getDetails(): String |

|  |
| --- |
| **SalariedEmployee** |
| - weeklySalary |
| SalariedEmployee(name: String, cnic: String, weeklySalary: float)  MonthlyEarnings(): float  getDetails(): String |

|  |
| --- |
| **HourlyEmployee** |
| - wage: float  - hours: float |
| HourlyEmployee(name: String, cnic: String, wage: float, hours: float)  MonthlyEarnings(): float  getDetails(): String |

Function details are given in the following table

|  |  |  |
| --- | --- | --- |
|  | MonthlyEarnings() | getDetails() |
| Employee | Abstract Method | Return name and cnic |
| Salaried Employee | Four times the weekly salary | Return name, cnic, and monthly earnings |
| Hourly Employee | Product of wages and hours.  For every extra hour above 40 hours, 1.5 times the wage | Return name, cnic, and monthly earnings |

Exercise

Use Python classes created in the previous lab of the above scenario to simulate an ***Employee Register*** for an Organization.

The system displays the main menu as follows

1. Add an Employee

A new employee is added to the Employee Register

User would be prompted to decide employee type and provide relevant details

1. Show Employee Details

User would be prompted to provide employee’s CNIC

If the record is available against the CNIC, it would be displayed along with the name, category, and the monthly earnings. Else, an appropriate message would be displayed

1. Terminate an Employee

User would be prompted to provide employee’s CNIC

If the record is available against the CNIC, the employee would be terminated and its record would be removed from the Employee Register. Else, an appropriate message would be displayed

1. Display Employee Register

A complete record of all the employees is displayed. The details to be included is the name, CNIC and the category of each employee

1. Exit the Application

**Code:**

from abc import ABC, abstractmethod

class Employee(ABC):

    def \_\_init\_\_(self, name, cnic):

        self.\_\_name = name

        self.\_\_cnic = cnic

    def getCnic(self):

        return self.\_\_cnic

    def getName(self):

        return self.\_\_name

    @abstractmethod

    def monthlySalary(self):

        pass

    def getDetails(self):

        return (self.\_\_name, self.\_\_cnic, self.monthlySalary())

class SalariedEmployee(Employee):

    def \_\_init\_\_(self, name, cnic, weeklySalary):

        super().\_\_init\_\_(name, cnic)

        self.\_\_weeklySalary = weeklySalary

    def monthlySalary(self):

        return self.\_\_weeklySalary \* 4

    def getDetails(self):

        return super().getDetails()

class HourlyEmployee(Employee):

    def \_\_init\_\_(self, name, cnic, wage, hours):

        super().\_\_init\_\_(name, cnic)

        self.\_\_wage = wage

        self.\_\_hours = hours

    def monthlySalary(self):

        if self.\_\_hours > 40:

            return 40 \* self.\_\_wage + (self.\_\_hours - 40) \* self.\_\_wage \* 1.5

        return self.\_\_wage \* self.\_\_hours

    def getDetails(self):

        return super().getDetails()

salaried = []

hourly = []

while True:

    print('1. Add an Employee')

    print('2. Show Employee Details')

    print('3. Terminate an Employee')

    print('4. Display Employee Register')

    print('5. Exit the Application')

    choice = int(input('Enter your choice: '))

    if choice == 1:

        print('1. Salaried Employee')

        print('2. Hourly Employee')

        choice = int(input('Enter your choice: '))

        if choice == 1:

            print()

            name = input('Enter name: ')

            cnic = input('Enter cnic: ')

            weeklySalary = int(input('Enter weekly salary: '))

            if cnic in [i.getCnic() for i in salaried] or cnic in [i.getCnic() for i in hourly]:

                print('Employee already exists')

            else:

                salaried.append(SalariedEmployee(name, cnic, weeklySalary))

                print('Salaried Employee added successfully')

                print()

        elif choice == 2:

            print()

            name = input('Enter name: ')

            cnic = input('Enter cnic: ')

            wage = int(input('Enter wage: '))

            hours = int(input('Enter hours: '))

            if cnic in [i.getCnic() for i in hourly] or cnic in [i.getCnic() for i in salaried]:

                print('Employee already exists')

            else:

                hourly.append(HourlyEmployee(name, cnic, wage, hours))

                print("Hourly Employee Added")

                print()

    elif choice == 2:

        print()

        cnic = input('Enter cnic: ')

        for emp in salaried:

            if emp.getCnic() == cnic:

                print(emp.getDetails())

                break

        print()

        for emp in hourly:

            if emp.getCnic() == cnic:

                print(emp.getDetails())

                break

        print()

    elif choice == 3:

        print()

        cnic = input('Enter cnic: ')

        if cnic in [i.getCnic() for i in salaried]:

            for emp in salaried:

                if emp.getCnic() == cnic:

                    salaried.remove(emp)

                    print('Employee terminated')

                    break

        elif cnic in [i.getCnic() for i in hourly]:

            print()

            for emp in hourly:

                if emp.getCnic() == cnic:

                    hourly.remove(emp)

                    print('Employee terminated')

                    break

            print()

        else:

            print('Employee not found')

            print()

    elif choice == 4:

        print()

        print('Salaried Employees:')

        print("Name","CNIC","Monthly Salary")

        for emp in salaried:

            print(emp.getDetails())

        print()

        print('Hourly Employees:')

        print("Name","CNIC","Monthly Salary")

        for emp in hourly:

            print(emp.getDetails())

        print()

    elif choice == 5:

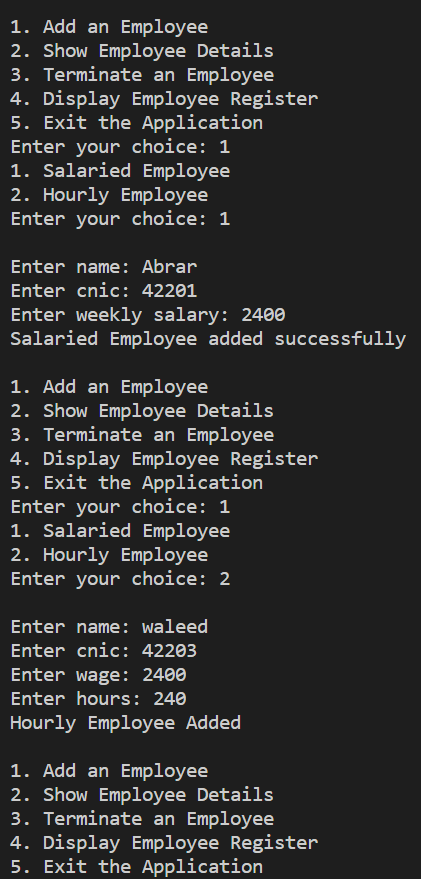
        break

    else:

        print('Invalid choice')

**Output:**

1. **Adding an Employee:**

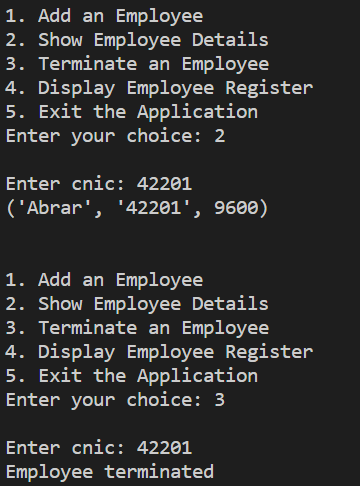
****

1. **Show Employee Details:**

**Text

Description automatically generated**

**3) Terminate an Employee:**

****

**4) Display Employee Register:**

**Text

Description automatically generated**

1. **Exit the Application:**

**Text

Description automatically generated**