Task:

Simple Real-Time Tasks on Polymorphism &

Abstraction and Polymorphism Tasks

1. Animals Speak Create a base class Animal with method sound(). Subclasses: - Dog → prints 'Bark' - Cat → prints 'Meow' . Create a list of animals and call sound() for each.

```
class Animal:
    def sound(self):
        pass
class Dog(Animal):
    def sound(self):
        print('Dog Barks')
class Cat(Animal):
    def sound(self):
        print('Cat sounds Meowww..')
animals=[Dog(),Cat()]
for animal in animals:
    animal.sound()
```

2. Calculator Operations Create a base class Calculator with method calculate(a, b). Subclasses: Add → adds two numbers - Subtract → subtracts two numbers Call calculate() for both.

```
class Calculator:
    def calculate(self,a,b):
        pass
class Add(Calculator):
    def calculate(self,a,b):
        print(a+b)
class Subtract(Calculator):
    def calculate(self,a,b):
```

```
        print(a-b)
p=Add()
p.calculate(10,20)
s=Subtract()
s.calculate(30,5)
```

3. Transport Ride Fare Class Transport with method fare(distance). Subclasses: - Bus → fare = distance * 10 - Train → fare = distance * 5 Show polymorphism by calling fare() with same distance.

```
class Transport:
    def fare(self,distance):
        pass
class Bus(Transport):
    def fare(self,distance):
        print(f'Bus fare from {distance}km is {distance*5}')
class Train(Transport):
    def fare(self,distance):
        print(f'Train fare from {distance}km is {distance*10}')
transports=[Bus(),Train()]
for transport in transports:
    transport.fare(15)
```

4. Shape Area Class Shape with method area(). Subclasses: - Square (side²) - Circle (π × r²)

Loop through objects and print areas.

```
class Shape:
    def area(self):
        pass
class Square(Shape):
    def area(self,side):
        print(f'Area of square for side {side} is {side*side}')
```

```
class Circle(Shape):
    def area(self,radius):
        pi=3.14
        print(f'Area of circle for radius {radius} is {pi*radius*radius} ')
shape=[Square(),Circle()]
for i in shape:
    i.area(10)
```

5. Employee Work Class Employee with method work(). Subclasses: - Developer → prints 'Writing code' - Tester → prints 'Testing software' Call work() on both employees.

```
class Employee:
    def work(self):
        pass
class Developer(Employee):
    def work(self):
        print('Writing code')
class Tester(Employee):
    def work(self):
        print('Testing Software')
employees=[Developer(),Tester()]
for employee in employees:
    employee.work()
```

Abstraction Tasks:

6. Vehicle Start Abstract class Vehicle with method start(). Subclasses: - Car → 'Car started' - Bike → 'Bike started'

```
from abc import ABC,abstractmethod
class Vehicle(ABC):
    @abstractmethod
```

```
    def start(self):
        pass
class Car(Vehicle):
    def start(self):
        print('Car Started')
class Bike(Vehicle):
    def start(self):
        print('Bike Started')
c1=Car()
c1.start()
b1=Bike()
b1.start()
```

7. Bank Account Abstract class BankAccount with method withdraw(amount). Subclasses: SavingsAccount → 'Withdrawn from savings' - CurrentAccount → 'Withdrawn from current'

```
from abc import ABC,abstractmethod
class BankAccount(ABC):
    @abstractmethod
    def withdraw(self,amount):
        pass
class SavingsAccount(BankAccount):
    def withdraw(self,amount):
        print(f'withdrawn {amount} from savings')
class CurrentAccount(BankAccount):
    def withdraw(self,amount):
        print(f'withdrawn {amount} from current')
withdrawn=[SavingsAccount(),CurrentAccount()]
for cash in withdrawn:
    cash.withdraw(1000)
```

8. Device Power Abstract class Device with method power_on(). Subclasses: - TV → 'TV is ON'

Laptop → 'Laptop is ON'

```
from abc import ABC,abstractmethod
class Device(ABC):
    @abstractmethod
    def power_on(self):
        pass
class TV(Device):
    def power_on(self):
        print('TV is ON')
class Laptop(Device):
    def power_on(self):
        print('Laptop is ON')
devices=[TV(),Laptop()]
for device in devices:
    device.power_on()
```

9. Student Exam Abstract class Exam with method start_exam(). Subclasses: - MathExam → 'Math exam started' - EnglishExam → 'English exam started'

```
from abc import ABC,abstractmethod
class Exam(ABC):
    @abstractmethod
    def start_exam(self):
        pass
class MathExam(Exam):
    def start_exam(self):
        print('Math exam started')
class EnglishExam(Exam):
    def start_exam(self):
```

```
        print('English exam started')
m1=MathExam()
m1.start_exam()
e1=EnglishExam()
e1.start_exam()
```

10. Report Generation Abstract class Report with method generate(). Subclasses: - PDFReport → 'PDF Report generated' - ExcelReport → 'Excel Report generated'

```
from abc import ABC,abstractmethod
class Report(ABC):
    @abstractmethod
    def generate(self):
        pass
class PDFReport(Report):
    def generate(self):
        print('PDF Report generated')
class ExcelReport(Report):
    def generate(self):
        print('Excel Report generated')
reports=[PDFReport(),ExcelReport()]
for report in reports:
    report.generate()
```