```python
#Create a person class with:
#i) two instance variable: name, age.
#ii) Create a parameterized constructor
#Create a student class. Inherit person class in Student class.
#Student class have:
#i) instance variable: rollno and stream.
#ii) Create a parameterized constructor to initialize all instance
variables of
#student class as well as Person class
#iii)Instance method: display() to print name, age, rollno and stream
#Create an object of Student class and call display method

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

    def display(self):
        print("Name: ",self.name)
        print("Age: ",self.age)

class Student(Person):
    def __init__(self,name,age,rollno,stream):
        super().__init__(name,age)
        self.rollno = rollno
        self.stream = stream

    def display(self):
        super().display()
        print("Roll No: ",self.rollno)
        print("Stream: ",self.stream)

s = Student("Saurabh",20,20045,"CSE")
s.display()
```

```
Name:  Saurabh
Age:  20
Roll No:  20045
Stream:  CSE
```

```python
#Write a Python class named Circle. Declare an instance variable,
radius and two methods that will compute the area and the perimeter of
a circle.


class Circle:
    def __init__(self,radius):
        self.radius = radius

    def area(self):
        return 3.14*(self.radius**2)

    def perimeter(self):
        return 2*3.14*self.radius


c = Circle(5)
print("Area of circle: ",c.area())
print("Perimeter of circle: ",c.perimeter())
```

```
Area of circle:  78.5
Perimeter of circle:  31.400000000000002
```

```python
#Write a Python program to create a calculator class. Include methods for basic arithmetic operations.

class Calculator:
    def __init__(self,num1,num2):
        self.num1 = num1
        self.num2 = num2

    def add(self):
        return self.num1 + self.num2

    def subtract(self):
        return self.num1 - self.num2

    def multiply(self):
        return self.num1 * self.num2

    def divide(self):
        return self.num1 / self.num2


c = Calculator(10,5)
print("Addition: ",c.add())
print("Subtraction: ",c.subtract())
print("Multiplication: ",c.multiply())
print("Division: ",c.divide())
```

```
Addition:  15
Subtraction:  5
Multiplication:  50
Division:  2.0
```

```python
#Write a Python program to create a class representing a shopping cart. Include methods for adding and removing items, and calculating the total price.

class Cart:
    def __init__(self):
        self.items = []

    def add_item(self, item):
        self.items.append(item)

    def remove_item(self, item):
        self.items.remove(item)

    def calculate_total(self):
        total = 0
        for item in self.items:
            total += item.price
        return total

class Item:
    def __init__(self, name, price):
        self.name = name
        self.price = price

cart = Cart()
cart.add_item(Item("Shirt", 20))
cart.add_item(Item("Pants", 30))
cart.add_item(Item("Shoes", 50))

total_price = cart.calculate_total()
print("Total price: ",total_price)
```

```
Total price:  100
```

```python
class Employee:
    def __init__(self, emp_name, emp_id, emp_salary, emp_department):
        self.emp_name = emp_name
        self.emp_id = emp_id
        self.emp_salary = emp_salary
        self.emp_department = emp_department

    def calculate_emp_salary(self, hours_worked):
        if hours_worked > 50:
            overtime = hours_worked - 50
            overtime_amount = overtime * (self.emp_salary / 50)
            return self.emp_salary + overtime_amount
        return self.emp_salary

    def emp_assign_department(self, new_department):
        self.emp_department = new_department

    def print_employee_details(self):
        print(f"Name: {self.emp_name}, ID: {self.emp_id}, Salary: 
{self.emp_salary}, Department: {self.emp_department}")
```

```python
emp1 = Employee("ADAMS", "E7876", 50000, "ACCOUNTING")
emp2 = Employee("JONES", "E7499", 45000, "RESEARCH")
emp3 = Employee("MARTIN", "E7900", 50000, "SALES")
emp4 = Employee("SMITH", "E7698", 55000, "OPERATIONS")


emp1.emp_assign_department("FINANCE")
emp1.print_employee_details()
salary_with_overtime = emp1.calculate_emp_salary(60)
print("Salary with overtime: ", salary_with_overtime)
emp2.emp_assign_department("SALES")
emp2.print_employee_details()
salary_with_overtime = emp2.calculate_emp_salary(60)
print("Salary with overtime: ", salary_with_overtime)
emp3.emp_assign_department("RESEARCH")
emp3.print_employee_details()
salary_with_overtime = emp3.calculate_emp_salary(60)
print("Salary with overtime: ", salary_with_overtime)
emp4.emp_assign_department("OPERATIONS")
emp4.print_employee_details()
salary_with_overtime = emp4.calculate_emp_salary(60)
print("Salary with overtime: ", salary_with_overtime)
```

```
Name: ADAMS, ID: E7876, Salary: 50000, Department: FINANCE
Salary with overtime:  60000.0
Name: JONES, ID: E7499, Salary: 45000, Department: SALES
Salary with overtime:  54000.0
Name: MARTIN, ID: E7900, Salary: 50000, Department: RESEARCH
Salary with overtime:  60000.0
Name: SMITH, ID: E7698, Salary: 55000, Department: OPERATIONS
Salary with overtime:  66000.0
```

```python
#Write a Python class BankAccount with attributes like account_number,
balance, date_of_opening and customer_name, and methods like deposit,
withdraw, and check_balance.


class BankAccount:
    def __init__(self, account_number, balance, date_of_opening,
customer_name):
        self.account_number = account_number
        self.balance = balance
        self.date_of_opening = date_of_opening
        self.customer_name = customer_name

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient balance")
        else:
            self.balance -= amount

    def check_balance(self):
        print("Balance: ", self.balance)


account = BankAccount("123456", 1000, "2023-01-01", "John Doe")
account.deposit(500)
account.withdraw(200)
account.check_balance()
```

```
Balance:  1300
```

```
#Create a class hierarchy for different types of geometric shapes,
including
#circles, rectangles, and triangles, using inheritance.
#Tasks:
#A. Define a base class called Shape with common attributes
#like colour and area.
#B. Implement subclasses for specific shape types such
#as Circle, Rectangle, and Triangle. Each subclass should inherit
#from the Shape class.
#C. Incorporate additional attributes and methods specific to each
#shape type. For example, a Circle class might have attributes
#like radius and methods like calculate_area.
#D. Use inheritance to create subclasses representing variations
within
#each shape type. For example, within the Rectangle class, create
#subclasses for Square and Parallelogram.
#E. Implement methods or attributes in the subclasses to demonstrate
#how inheritance allows for the sharing of attributes and methods
#from parent classes.
#F. Create instances of the various shape classes and test their
#functionality to ensure that attributes and methods work as
#expected.


class Shape:  # Base class
    def __init__(self, color):
        self.color = color

    def area(self):
        pass
```

```python
class Circle(Shape):
    def __init__(self, radius, color):
        super().__init__(color)
        self.radius = radius

    def area(self):
        return 3.14 * (self.radius ** 2)

class Rectangle(Shape):
    def __init__(self, width, height, color):
        super().__init__(color)
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

class Square(Rectangle):
    def __init__(self, side, color):
        super().__init__(side, side, color)

class Parallelogram(Rectangle):
    def __init__(self, base, height, color):
        super().__init__(base, height, color)

class Triangle(Shape):
    def __init__(self, base, height, color):
        super().__init__(color)
        self.base = base
        self.height = height

    def area(self):
        return 0.5 * self.base * self.height


circle = Circle(5, "red")
rectangle = Rectangle(4, 6, "blue")
square = Square(4, "green")
parallelogram = Parallelogram(5, 3, "yellow")
triangle = Triangle(4, 5, "purple")

print("Circle Area:", circle.area())  # ... existing code ...
print("Rectangle Area:", rectangle.area())  # ... existing code ...
print("Square Area:", square.area())  # ... existing code ...
print("Parallelogram Area:", parallelogram.area())  # ... existing
code ...
print("Triangle Area:", triangle.area())  # ... existing code ...
```

```
Circle Area: 78.5
Rectangle Area: 24
```

```
Square Area: 16
Parallelogram Area: 15
Triangle Area: 10.0

#WAP to find the number of words in the given text file
#Hints:
#Use the split() method to separate words.


with open("text.txt", "r") as file:
    text = file.read()
words = text.split()
print("Number of words: ", len(words))

Number of words:  2

#Write a program to write “Happy Programming” in a text file and read
it

with open("text.txt", "w") as file:
    text = file.write("Hello Programming")

with open("text.txt", "r") as file:
    text = file.read()
print(text)




Hello Programming

#.WAP to demonstrate the working of the following functions:
#i) read()
#ii) read(n)
#iii)readline()
#iv) readlines()


with open("text.txt", "r") as file:
    text = file.read()

print(text)


with open("text.txt", "r") as file:
    text = file.read(5)

print(text)


with open("text.txt", "r") as file:
    text = file.readline()
```

```
print(text)


with open("text.txt", "r") as file:
    text = file.readlines()

print(text)
```

```
Hello Programming
Hello
Hello Programming
['Hello Programming']
```

```
#WAP that exhibits the working of the following functions:
#i. write()
#ii. writelines()


with open("text.txt", "w") as file:
    text = file.write("Hello Programming")

with open("text.txt", "a") as file:
    text = file.write("Happy Programming")

with open("text.txt", "r") as file:
    text = file.read()
print(text)
```

```
Hello ProgrammingHappy Programming
```

```
#Write a Python program to read first n lines of a file.

with open("text.txt", "r") as file:
    text = file.readlines()

print(text)
```

```
['Hello ProgrammingHappy Programming']
```

```
#Write a Python program to append text to a file and display the text.


with open("text.txt", "a") as file:
    text = file.write("Happy Programming")

with open("text.txt", "r") as file:
    text = file.read()
print(text)
```

```
Hello ProgrammingHappy ProgrammingHappy Programming
```

```
#Write a Python program to read last n lines of a file.

with open("text.txt", "r") as file:
    text = file.readlines()

print(text)
```

```
['Hello ProgrammingHappy ProgrammingHappy Programming']
```

```
#Write a Python program to read a file line by line and store it into
a list.

with open("text.txt", "r") as file:
    text = file.readlines()

print(text)
```

```
['Hello ProgrammingHappy ProgrammingHappy Programming']
```

```
#Write a program to exhibit these concepts:
#i. try
#ii. except
#iii. finally


try:
    result = 10 / 0
except ZeroDivisionError:
    print("Division by zero is not allowed")
finally:
    print("Execution completed")
```

```
Division by zero is not allowed
Execution completed
```

```
#Write a Python program to handle a ZeroDivisionError exception when
dividing a number by zero.

try:
    result = 10 / 0
except ZeroDivisionError:
    print("Division by zero is not allowed")
finally:
    print("Execution completed")
```

```
Division by zero is not allowed
Execution completed
```

```
#Write a Python program that prompts the user to input an integer and
raises a  ValueError exception if the input is not a valid integer.
```

```python
try:
    num = int(input("Enter an integer: "))
except ValueError:
    print("Invalid input. Please enter an integer.")
```

```
Invalid input. Please enter an integer.
```

```python
#WAP that exhibits multiple except blocks along with default block


try:
    num = int(input("Enter an integer: "))
except ValueError:
    print("Invalid input. Please enter an integer.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
```

```
Invalid input. Please enter an integer.
```

```python
#WAP that exhibits except blocks that can catch multiple exceptions.

try:
    num = int(input("Enter an integer: "))
except (ValueError, ZeroDivisionError):
    print("Invalid input. Please enter an integer.")
except Exception as e:
    print("An error occurred: ", e)
```

```
Invalid input. Please enter an integer.
```

```python
#WAP to demonstrate how to use lambda in map() function.

numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
print(squared)
```

```
[1, 4, 9, 16, 25]
```

```python
#WAP to demonstrate how to use lambda in filter() function.

numbers = [1, 2, 3, 4, 5]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)
```

```
[2, 4]
```

```python
#Write a Python program to filter a list of integers into list of even
numbers  and list of odd numbers using Lambda. [Hint: use lambda in
filter() ]

numbers = [1, 2, 3, 4, 5]
```

```python
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
odd_numbers = list(filter(lambda x: x % 2 != 0, numbers))
print("Even numbers:", even_numbers)
print("Odd numbers:", odd_numbers)
```

```
Even numbers: [2, 4]
Odd numbers: [1, 3, 5]
```

```python
#Write a Python program to square and cube every number in a given
list of  integers using Lambda. [Hint: use lambda in map() ].

numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x**2, numbers))
cube_numbers = list(map(lambda x: x**3, numbers))
print("Squared numbers:", squared_numbers)
print("Cube numbers:", cube_numbers)
```

```
Squared numbers: [1, 4, 9, 16, 25]
Cube numbers: [1, 8, 27, 64, 125]
```

```python
#Write a Python program to create a lambda function that adds 15 to a
given  number passed in as an argument.

add_15 = lambda x: x + 15
result = add_15(10)
print("Result:", result)
```

```
Result: 25
```

```python
#Create a lambda function that multiplies argument x with argument y
and  prints the resul


multiply = lambda x, y: x * y
result = multiply(5, 6)
print("Result:", result)
```

```
Result: 30
```