python-assignment-3

November 9, 2024

[1]:
```python
#1 Create a person class with:
#i) two instance variable: name, age.
#ii) Create a parameterized constructor

class person:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def display_info(self):
        print(f"Name:{self.name} \nAge:{self.age}")

#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 student(person):
    def __init__(self,name,age,roll_no,stream):
        super().__init__(name,age)
        self.roll_no = roll_no
        self.stream = stream
    def display_student_info(self):
        self.display_info()
        print(f"Roll No.:{self.roll_no} \nStream:{self.stream}")
student1 = student("Alice",16,21,"Science")
student1.display_student_info()
```

```
Name:Alice
Age:16
Roll No.:21
Stream:Science
```

[2]:
```python
#2 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.
```

```
import math
class circle:
    def __init__(self,radius):
        self.radius = radius

    def area(self):
        return math.pi*self.radius**2

    def perimeter(self):
        return 2*math.pi*self.radius
circle1 = circle(7)
print(f"Area:{circle1.area()}\nPerimeter:{circle1.perimeter()}")
```

```
Area:153.93804002589985
Perimeter:43.982297150257104
```

[3]:
```
#3 Write a Python program to create a calculator class. Include methods for␣
 ↪basic arithmetic operations.
# IMPORTANT:-
# We have use constructor(__init__) in which we have defined 'a' and 'b' with␣
 ↪'self.a' and 'self.b' that is why we need write (return self.a+self.b) etc..␣
 ↪but if we have not used constructor then we could have directly wrote␣
 ↪(return a+b)
#We don't need to use import module in this case

class calculator:
    def __init__(self,a,b):
        self.a = a
        self.b = b
    def add(self):
        return self.a+self.b
    def sub(self):
        return self.a-self.b
    def mul(self):
        return self.a*self.b
    def div(self):
        if self.b==0:
            return "Error: Division by Zero is undefined"
        return self.a/self.b

calculator1 = calculator(12,10)
print(f"Addition:{calculator1.add()}\nSubstraction:{calculator1.
 ↪sub()}\nMultiplication:{calculator1.mul()}\nDivision:{calculator1.div()}")
```

```
Addition:22
Substraction:2
```

```
Multiplication:120
Division:1.2
```

[4]:
```python
#4 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 ShoppingCart:
    def __init__(self):
        self.items = []
    def add_items(self,item_name,price):
        self.items.append({"Name": item_name,"Price": price})
        print(f"Added {item_name} in the cart of ${price}")
    def remove_items(self,item_name):
        for item in self.items:
            if item["Name"]==item_name:
                self.items.remove(item)
                print(f"Removed {item_name} from the cart")
                return
        print("Item not found")
    def calculate_total(self):
        total = sum(item["Price"]for item in self.items)
        return total
    def display_cart(self):
        if not self.items:
            print("The cart is empty")
        else:
            print("Items in the cart are:")
            for item in self.items:
                print(f"{item["Name"]}:${item["Price"]}")
            print(f"total:{self.calculate_total()}")
cart = ShoppingCart()
cart.add_items("Apple",2)
cart.add_items("Mango",5)
cart.display_cart()
cart.remove_items("Mango")
cart.display_cart()
```

```
Added Apple in the cart of $2
Added Mango in the cart of $5
Items in the cart are:
Apple:$2
Mango:$5
total:7
Removed Mango from the cart
Items in the cart are:
Apple:$2
total:2
```

[5]:
```
#5 Write a Python class Employee with attributes like emp_id,␣
 ↪emp_name,emp_salary, and emp_department and methods like␣
 ↪calculate_emp_salary,emp_assign_department, and print_employee_details.
#Sample Employee Data:
#"ADAMS", "E7876", 50000, "ACCOUNTING"
#"JONES", "E7499", 45000, "RESEARCH"
#"MARTIN", "E7900", 50000, "SALES"
#"SMITH", "E7698", 55000, "OPERATIONS"
class Employee:
    def __init__(self,emp_id,emp_name,emp_salary,emp_department=None):
        self.emp_id = emp_id
        self.emp_name = emp_name
        self.emp_salary = emp_salary
        self.emp_department = emp_department
    def calculate_emp_salary(self,hours_worked):
        if hours_worked>50:
            salary = self.emp_salary/50
            overtime = hours_worked-50
            overtime_amount= (overtime*(salary))
            total_salary = self.emp_salary+overtime_amount
        else:
            total_salary = self.emp_salary
        self.emp_salary = total_salary
        print(f"Total Salary (with overtime if applicable): ${total_salary:.
 ↪2f}")
    def emp_assign_department(self,new_department):
        self.emp_department = new_department
        print(f"{self.emp_name} has been assigned to {self.emp_department}␣
 ↪department")
    def display_employee_details(self):
        print(f"Employee ID:{self.emp_id}")
        print(f"Employee Name:{self.emp_name}")
        print(f"Employee salary:{self.emp_salary}")
        print(f"Employee Department:{self.emp_department}")
a = Employee(emp_name="ADAMS", emp_id="E7876", emp_salary=50000,␣
 ↪emp_department="ACCOUNTING")
a.display_employee_details()
a.emp_assign_department("SALES")
a.calculate_emp_salary(0)
a.display_employee_details()
a.calculate_emp_salary(55)
a.display_employee_details()
```

```
Employee ID:E7876
Employee Name:ADAMS
Employee salary:50000
Employee Department:ACCOUNTING
```

```
ADAMS has been assigned to SALES department
Total Salary (with overtime if applicable): $50000.00
Employee ID:E7876
Employee Name:ADAMS
Employee salary:50000
Employee Department:SALES
Total Salary (with overtime if applicable): $55000.00
Employee ID:E7876
Employee Name:ADAMS
Employee salary:55000.0
Employee Department:SALES
```

[6]:
```python
#6 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):
        if amount>0:
            self.balance += amount
            print(f"${amount} deposited sucessfully")
        else:
            print(f"Deposit amount must be positive")
    def withdraw(self,amount):
        if amount>0:
            if amount<=self.balance:
                self.balance-=amount
                print(f"{amount} withdrawn succesfully")
            else:
                print("Insufficent balance")
        else:
            print("Withdrwal amount must be positive")
    def check_balance(self):
        print(f"Current balance: ${self.balance}")
    def display_BankAccount_details(self):
        print(f"Account Number:{self.account_number}\nBalance:{self.
 ↪balance}\nDate of opening:{self.date_of_opening}\nCustomer Name:{self.
 ↪customer_name}" )
account =␣
 ↪BankAccount(account_number=198919782560,balance=300000,date_of_opening="1980-09-19",custome
account.display_BankAccount_details()
account.deposit(11000)
account.withdraw(20000)
```

```
account.check_balance()
account.display_BankAccount_details()
```

```
Account Number:198919782560
Balance:300000
Date of opening:1980-09-19
Customer Name:Rishi
$11000 deposited sucessfully
20000 withdrawn succesfully
Current balance: $291000
Account Number:198919782560
Balance:291000
Date of opening:1980-09-19
Customer Name:Rishi
```

[7]:
```
#7 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.

import math
#A. Define a Parent or Base class
class Shapes:
    def __init__(self,colour="Transperant"):
        self.colour = colour
        self.area = 0 #Area is set up to zero
    def display_info(self):
        #Display the shape's colour and area
        print(f"Colour: {self.colour}\nArea: {self.area}")

#B. Implement subclasses for specific shaps
#Circle subclass ineriting from shapes
class Circle(Shapes):
```

```
    def __init__(self,radius,colour = "Transperent"):
        super().__init__(colour) #calls the shape's constructor with colour of␣
 ↪the shape
        self.radius = radius
    def calculate_area(self):
        self.area = math.pi*(self.radius**2)

class Rectangle(Shapes):
    def __init__(self,length,breadth,colour="Transperant"):
        super().__init__(colour)
        self.length = length
        self.breadth = breadth
    def calculate_area(self):
        self.area = self.length*self.breadth

class Triangle(Shapes):
    def __init__(self,base,height,colour="Transperant"):
        super().__init__(colour)
        self.base = base
        self.height = height
    def calculate_area(self):
        self.area = self.base*self.height*0.5

class Square(Rectangle):
    def __init__(self,side,colour="Transperant"):
        super().__init__(side,side,colour)

class Parallelogram(Rectangle):
    def __init__(self,base,height,colour="Transperant"):
        super().__init__(base,height,colour)

circle = Circle(radius=4,colour="Orange")
circle.calculate_area()
circle.display_info()

rectangle = Rectangle(length=12,breadth=10,colour="Blue")
rectangle.calculate_area()
rectangle.display_info()

triangle = Triangle(base=11,height=2,colour="Brown")
triangle.calculate_area()
triangle.display_info()

square = Square(side=11,colour="White")
square.calculate_area()
square.display_info()
```

```
parallelogram = Parallelogram(base=20,height=5,colour="Red")
parallelogram.calculate_area()
parallelogram.display_info()
```

```
Colour: Orange
Area: 50.26548245743669
Colour: Blue
Area: 120
Colour: Brown
Area: 11.0
Colour: White
Area: 121
Colour: Red
Area: 100
```

[8]:
```
#8 WAP to find the number of words in the given text file
def count_words_in_file(filename):
    try:
        with open(filename,'r') as file:
            content = file.read()
            words = content.split()
            words_count = len(words)
            print(f"The number of words in {filename} is:{words_count}")
    except FileNotFoundError:
        print(f"Error: The file {filename} was not found.")

filename = 'sample.txt'
count_words_in_file(filename)
```

```
Error: The file sample.txt was not found.
```

[9]:
```
#9 Write a program to write "Happy Programming" in a text file and read it
with open("sample.txt",'w+') as f:
    a = "Happy Programming"
    f.write(a)
    f.seek(0)
    print(f.read())
```

```
Happy Programming
```

[10]:
```
#10 WAP to demonstrate the working of the following functions:
#i) read()
#ii) read(n)
#iii)readline()
#iv) readlines()

with open("Sample.txt",'w') as f:
```

```
    f.write("Hello, World!\n")
    f.write("This is a Sample text file.\n")
    f.write("It conatins multiple lines.\n")
    f.write("Each line demonstrate different read function.")

with open("sample.txt",'r') as f:
    #1 Using read()
    print("Using read()")
    a = f.read()
    print(a)
    print("------------------")

    f.seek(0)
    #2 Using read(n)
    print("Using read(n)")
    b = f.read(15)
    print(b)
    print("------------------")

    f.seek(0)
    #3 Using readline()
    print("Using readline()")
    line1 = f.readline()
    print(line1,end="")
    line2 = f.readline()
    print(line2,end="")
    print("------------------")

    f.seek(0)
    #4 Using readlines()
    print("Using readlines()")
    lines = f.readlines()
    print(lines)
    print("------------------")
```

```
Using read()
Hello, World!
This is a Sample text file.
It conatins multiple lines.
Each line demonstrate different read function.
------------------
Using read(n)
Hello, World!
T
------------------
Using readline()
Hello, World!
```

```
This is a Sample text file.
------------------
Using readlines()
['Hello, World!\n', 'This is a Sample text file.\n', 'It conatins multiple
lines.\n', 'Each line demonstrate different read function.']
-----------------
```

[11]:
```python
#11 11.WAP that exhibits the working of the following functions:
#i. write()
#ii. writelines()
with open("t_write.txt",'w') as f:
    a = "Hello, Miss Shweta"
    f.write(a)
    f.write("\nThis is another line.")
print("Content written using write()")

with open("t_writelines.txt",'w') as f:
    lines = ["First line: This is the first line.\n","Second line: This is the␣
 ↪second line.\n","Third line: This is the third line.\n"]
    f.writelines(lines)
print("Content written using writelines()")
```

```
Content written using write()
Content written using writelines()
```

[12]:
```python
#12 Write a Python program to read first n lines of a file.
def read_first_n_lines(filename,n):
    try:
        with open(filename,'r') as f:
            for line in range(n):
                line = f.readline()
                if line=='':
                    break
                print(line,end="")
    except FileNotFoundError:
        print(f"Error: The file {filename} was not found.")

read_first_n_lines("sample.txt",2)
```

```
Hello, World!
This is a Sample text file.
```

[13]:
```python
#13 Write a Python program to append text to a file and display the text.
text_to_append = "\nThis is the appended text."
with open("sample.txt",'a+') as f:
    f.write(text_to_append)
    f.seek(0)
    print(f.read())
```

```
Hello, World!
This is a Sample text file.
It conatins multiple lines.
Each line demonstrate different read function.
This is the appended text.
```

[14]:
```python
#14 Write a Python program to read last n lines of a file.
def read_last_n_lines(filename,n):
    try:
        with open(filename,'r') as f:
            lines = f.readlines()
            last_n_lines = lines[-n:]
            for line in last_n_lines:
                print(line,end="")
    except FileNotFoundError:
        print(f"Error: The file {filename} was not found")

read_last_n_lines('sample.txt',4)
```

```
This is a Sample text file.
It conatins multiple lines.
Each line demonstrate different read function.
This is the appended text.
```

[15]:
```python
#15 Write a Python program to read a file line by line and store it into a list.
def files_to_list(filename):
    lines=[]
    try:
        with open(filename,'r') as f:
            for line in f:
                lines.append(line.strip())
        return lines
    except FileNotFoundError:
        print(f"The file {filename} was not found")
        return[]
print("Files content as list:")
t = files_to_list("sample.txt")
print(t)
```

```
Files content as list:
['Hello, World!', 'This is a Sample text file.', 'It conatins multiple lines.',
'Each line demonstrate different read function.', 'This is the appended text.']
```

[16]:
```python
#16 Write a program to exhibit these concepts:
#i. try
#ii. except
#iii. finally
```

```
def divide_numbers(a,b):
    try:
        result=a/b
        print(f"The result when we divide {a} by {b} is {result}")
    except ZeroDivisionError:
        print("Error: Division by zero is not allowed")
    finally:
        print("Execution of divide_numbers function is completed")
divide_numbers(12,6)
divide_numbers(10,0)
```

```
The result when we divide 12 by 6 is 2.0
Execution of divide_numbers function is completed
Error: Division by zero is not allowed
Execution of divide_numbers function is completed
```

[17]:
```
#17 Write a Python program to handle a ZeroDivisionError exception when␣
 ↪dividing a number by zero.
def divide_numbers(a,b):
    try:
        result = a/b
        print(f"The result of division of {a} by {b} is {result}")
    except ZeroDivisionError:
        print("Error: Division by zero is not allowed ")
divide_numbers(5,0)
```

```
Error: Division by zero is not allowed
```

[18]:
```
#18.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.
def input_integer():
    try:
        a = int(input("Enter an value:"))
        print(f"You entered the integer: {a}")
    except ValueError:
        print("Error: This is not a valid integer. Enter a valid integer")
input_integer()
```

```
You entered the integer: 13
```

[19]:
```
#19.WAP that exhibits multiple except blocks along with default block
try:
    x,y = input("Enter two numbers:").split()
    x = int(x)
    y = int(y)
    print(x/y)
except ZeroDivisionError as msg:
```

```
    print("y shoul not be zero",msg)
except ValueError as msg:
    print("Type inappropriate",msg)
except:
    print("Other Errors")
```

```
1.8571428571428572
```

[20]:
```
#20.WAP that exhibits except blocks that can catch multiple exceptions.
try:
    a,b=input("Enter two integer:").split()
    a = int(a)
    b = int(b)
    print(a/b)
except (ZeroDivisionError,ValueError) as msg:
    print(msg)
```

```
1.8571428571428572
```

[21]:
```
#21.WAP to demonstrate how to use lambda in map() function.
L = [1,2,3,4,5]
doubler = map(lambda x:x*2,L)
print(list(doubler))
```

```
[2, 4, 6, 8, 10]
```

[22]:
```
#22.WAP to demonstrate how to use lambda in filter() function.
age = [13,5,19,40,21]
adults = filter(lambda x:x>18,age)
print(list(adults))
```

```
[19, 40, 21]
```

[23]:
```
#23.Write a Python program to filter a list of integers into list of even␣
 ↪numbers and list of odd numbers using Lambda.
integers = [12,3,2,9,7,5,4]
even_number = list(filter(lambda x:x%2==0,integers))
odd_number = list(filter(lambda x:x%2!=0,integers))
print(integers)
print("Odd number:",odd_number)
print("Even number:",even_number)
```

```
[12, 3, 2, 9, 7, 5, 4]
Odd number: [3, 9, 7, 5]
Even number: [12, 2, 4]
```

[24]:
```
#24.Write a Python program to square and cube every number in a given list of␣
 ↪integers using Lambda.
```

```
integers = [1,5,9,12,15]
Square_number = list(map(lambda x:x**2,integers))
Cube_number = list(map(lambda x:x**3,integers))
print("Square of every number in the list integers is:",Square_number)
print("Cube of every number in the list integers is:",Cube_number)
```

```
Square of every number in the list integers is: [1, 25, 81, 144, 225]
Cube of every number in the list integers is: [1, 125, 729, 1728, 3375]
```

[25]:
```
#25.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
num = int(input("Enter a number:"))
result = add_15(num)
print(f"{num}+15 = {result}")
```

```
13+15 = 28
```

[26]:
```
#26.Create a lambda function that multiplies argument x with argument y and␣
 ↪prints the result.
x,y = input("Enter two integers:").split()
x = int(x)
y = int(y)
multiply_x_y = lambda x,y:x*y
n = multiply_x_y(x,y)
print(f"{x}*{y}={n}")
```

```
13*31=403
```

[ ]: