Name : Alisina Moradi F\Name : Mohammad Reza Department : Information Technology

Basic Class and Object Exercises

1)Create a class called Person with attributes name and age. Create an object of the class and print its attributes.

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

person = Person("Alice", 30)
print(person.name, person.age)
```

2)Add a method called greet to the Person class that prints a greeting message including the person's name.

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

    def greet(self):
        print(f"Hello, my name is {self.name}.")

person = Person("Alice", 30)
person.greet()
```

3)Create a class called Car with attributes make, model, and year. Include a method to print out the car's details.

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def print_details(self):
        print(f"Car details: {self.year} {self.make} {self.model}")

car = Car("Toyota", "Corolla", 2020)
car.print_details()
```

4)Create a class Circle with a method to compute the area. Initialize the class with the radius.

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

    def compute_area(self):
        return 3.14159 * self.radius * self.radius

circle = Circle(5)
print(circle.compute_area())
```

5)Create a class Rectangle with methods to compute the area and perimeter. Initialize the class with the length and width.

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def compute_area(self):
        return self.length * self.width

    def compute_perimeter(self):
        return 2 * (self.length + self.width)

rectangle = Rectangle(4, 5)
print(rectangle.compute_area())
```

```
print(rectangle.compute_perimeter())
```

Inheritance and Polymorphism Exercises

6)Create a base class Animal with a method speak. Create two derived classes Dog and Cat that override the speak method.

```
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "Woof"

class Cat(Animal):
    def speak(self):
        return "Meow"

dog = Dog()
cat = Cat()
print(dog.speak())
print(cat.speak())
```

7)Create a base class Shape with a method area. Create derived classes Square and Triangle that implement the area method.

```
class Shape:
    def area(self):
        pass

class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side * self.side

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

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

square = Square(4)
triangle = Triangle(3, 6)
print(square.area())
print(triangle.area())
```

8)Create a class Employee with attributes name and salary. Create a derived class Manager with an additional attribute department.

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

class Manager(Employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

manager = Manager("Alice", 80000, "HR")
print(manager.name, manager.salary, manager.department)
```

9)Create a base class Vehicle with a method drive. Create derived classes Bike and Truck that override the drive method.

```
class Vehicle:
    def drive(self):
        pass
```

```
class Bike(Vehicle):
    def drive(self):
        return "Bike is driving"

class Truck(Vehicle):
    def drive(self):
        return "Truck is driving"

bike = Bike()
truck = Truck()
print(bike.drive())
print(truck.drive())
```

10)Create a base class Bird with a method fly.
Create derived classes Eagle and Penguin. Override the fly method in Penguin to indicate that penguins cannot fly.

```
class Bird:
    def fly(self):
        pass

class Eagle(Bird):
    def fly(self):
        return "Eagle is flying"

class Penguin(Bird):
    def fly(self):
        return "Penguins cannot fly"

eagle = Eagle()
penguin = Penguin()
print(eagle.fly())
print(penguin.fly())
```

EncapsulationandAbstractionExercises

11)Create a class Account with private attributes balance. Provide public methods to deposit and withdraw money.

```
class Account:
    def __init__(self, balance):
        self.__balance = balance

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

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount

    def get_balance(self):
        return self.__balance

account = Account(1000)
account.deposit(500)
account.withdraw(200)
print(account.get_balance())
```

12)Create a class Book with private attributes title, author, and pages. Provide public methods to get and set these attributes.

```
class Book:
    def __init__(self, title, author, pages):
        self.__title = title
        self.__author = author
        self.__pages = pages

    def get_title(self):
        return self.__title
```

```
    def get_author(self):
        return self.__author

    def get_pages(self):
        return self.__pages

    def set_title(self, title):
        self.__title = title

    def set_author(self, author):
        self.__author = author

    def set_pages(self, pages):
        self.__pages = pages

book = Book("1984", "George Orwell", 328)
print(book.get_title())
print(book.get_author())
print(book.get_pages())
```

13)Create a class Laptop with private attributes brand, model, and price. Provide a method to apply a discount and a method to display laptop details.

```
class Laptop:
    def __init__(self, brand, model, price):
        self.__brand = brand
        self.__model = model
        self.__price = price

    def apply_discount(self, discount):
        self.__price -= discount

    def display_details(self):
        return f"Laptop: {self.__brand} {self.__model}, Price: {self.__price}"

laptop = Laptop("Dell", "XPS 15", 1500)
laptop.apply_discount(200)
print(laptop.display_details())
```

14)Create a class BankAccount with private attributes account_number and balance. Provide methods to deposit, withdraw, and check the balance.

```
class BankAccount:
    def __init__(self, account_number, balance):
        self.__account_number = account_number
        self.__balance = balance

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

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount

    def check_balance(self):
        return self.__balance

bank_account = BankAccount("123456789", 1000)
bank_account.deposit(500)
bank_account.withdraw(300)
print(bank_account.check_balance())
```

15)Create a class Student with private attributes name, grade, and age. Provide methods to get and set these attributes and a method to display the student's details.

```
class Student:
    def __init__(self, name, grade, age):
        self.__name = name
        self.__grade = grade
```

```
        self.__age = age

    def get_name(self):
        return self.__name

    def get_grade(self):
        return self.__grade

    def get_age(self):
        return self.__age

    def set_name(self, name):
        self.__name = name

    def set_grade(self, grade):
        self.__grade = grade

    def set_age(self, age):
        self.__age = age

    def display_details(self):
        return f"Name: {self.__name}, Grade: {self.__grade}, Age: {self.__age}"

student = Student("John", "A", 20)
print(student.display_details())
```

Class Relationships and Advanced Concepts Exercises

16. Create a class Library with attributes name and books (a list of Book objects). Provide methods to add and remove books.

```
class Book:
    def __init__(self,title,author):
        self.title = title
        self.author = author
class Library:
    def __init__(self,name):
        self.name = name
        self.books = []
    def add_books(self,book):
        self.books.append(book)
    def remove_books(self,book):
        self.books.remove(book)

library = Library("My Library")
book1 = Book("kabul","Alisina")
book2 = Book("Mmountain","Ahmad")

library.add_books(book1)
library.add_books(book2)
print([ book.title for book in library.books])
library.remove_books(book1)
print([ book.title for book in library.books])
```

17. Create a class School with attributes name and students (a list of Student objects). Provide methods to add and remove students.

```
class Student:
    def __init__(self,name,grade):
        self.name = name
        self.grade = grade
class School:
    def __init__(self,name):
        self.name = name
        self.students = []
    def add_students(self,student):
        self.students.append(student)
    def remove_students(self,student):
        self.students.remove(student)
```

```
school = School("Moradi")
student1 = Student("alisina",12)
student2= Student("Ahmad",10)

school.add_students(student1)
school.add_students(student2)
print([ student.name for student in school.students])
school.remove_students(student1)
print([ student.name for student in school.students])
```

18. Create a class Team with attributes name and members (a list of Person objects). Provide methods to add and remove members.

```
class Person:
    def __init__(self,name,grade):
        self.name = name
        self.grade = grade
class Team:
    def __init__(self,name):
        self.name = name
        self.persons = []
    def add_persons(self,person):
        self.persons.append(person)
    def remove_persons(self,person):
        self.persons.remove(person)

team = Team("Moradi")
person1 = Person("alisina",12)
person2= Person("Ahmad",10)

team.add_persons(person1)
team.add_persons(person2)
print([ person.name for person in team.persons])
team.remove_persons(person1)
print([ person.name for person in team.persons])
```

19. Create a class Company with attributes name and employees (a list of Employee objects). Provide methods to add and remove employees.

```
class Emploee:
    def __init__(self,name,grade):
        self.name = name
        self.grade = grade
class Company:
    def __init__(self,name):
        self.name = name
        self.emploees = []
    def add_emploees(self,emploee):
        self.emploees.append(emploee)
    def remove_emploees(self,emploee):
        self.emploees.remove(emploee)

company = Company("Moradi")
emploee1 = Emploee("alisina",12)
emploee2= Emploee("Ahmad",10)

company.add_emploees(emploee1)
company.add_emploees(emploee2)
print([ emploee.name for emploee in company.emploees])
company.remove_emploees(emploee1)
print([ emploee.name for emploee in company.emploees])
```

20. Create a class Zoo with attributes name and animals (a list of Animal objects). Provide methods to add and remove animals.

```
class Animal:
    def __init__(self,name,grade):
        self.name = name
        self.grade = grade
```

```
class Zoo:
    def __init__(self,name):
        self.name = name
        self.animals = []
    def add_animals(self,animal):
        self.animals.append(animal)
    def remove_animals(self,animal):
        self.animals.remove(animal)

zoo = Zoo("Nora")
animal1 = Animal("zebra",12)
animal2= Animal("lion",10)

zoo.add_animals(animal1)
zoo.add_animals(animal2)
print([animal.name for animal in zoo.animals])
zoo.remove_animals(animal1)
print([animal.name for animal in zoo.animals])
```

File Handling and Exceptions Exercises

21. Create a class FileManager with methods to read from and write to a file.

```
class FileManager:
    @staticmethod
    def write_file(content):
        with open("mahmod.text", 'w') as file:
            file.write(content)

    @staticmethod
    def read_file(file_path):
        with open(file_path, 'r') as file:
            return file.read()

# Example usage
#file_manager = FileManager()
FileManager.write_file('Hello, i am good  and World!')
print(FileManager.read_file('mahmod.text'))
```

22. Create a class Log with methods to write error messages to a log file.

```
import datetime
class Log:
    @staticmethod
    def write_file(massage):
        with open("reza.text","a") as file:
            file.write(f"{datetime.datetime.now()}-ERROR:{massage}\n")

    @staticmethod
    def read_file(file_path):
        with open(file_path,"r") as file:
            return file.read()

Log.write_file("An error ocured ok")
print(Log.read_file("reza.text"))
```

23. Create a class Config that reads configuration settings from a file and provides methods to access these settings.

```
class Config:
    def __init__(self, filename):
        self.settings = {}
        self.load_config(filename)

    def load_config(self, filename):
        with open(filename, 'r') as file:
            for line in file:
                key, value = line.strip().split("=")
```

```
            self.settings[key.strip()] = value.strip()

    def get(self, key, default=None):
        return self.settings.get(key, default)

config = Config('config.txt')
print(config.get('name'))    # خروجی: alisina
print(config.get('grade'))   # خروجی: 12
print(config.get('address')) # خروجی: barchi
```

24. Create a class Database that connects to a database and provides methods to execute queries. Handle exceptions if the connection fails.

```
import sqlite3
class Database:
    def __init__(self, db_name):
        self.db_name = db_name
        self.connection = None

    def connect(self):
        try:
            self.connection = sqlite3.connect(self.db_name)
        except sqlite3.Error as e:
            print(f"Connection failed: {e}")

    def execute_query(self, query):
        if self.connection:
            try:
                cursor = self.connection.cursor()
                cursor.execute(query)
                self.connection.commit()
            except sqlite3.Error as e:
                print(f"Query failed: {e}")

# Example usage
db = Database('example.db')
db.connect()
db.execute_query('CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)')
```

25. Create a class Report that generates a report from data in a file. Provide methods to handle exceptions if the file does not exist or cannot be read.

```
class Report:
    @staticmethod
    def generate_report(file_path):
        try:
            with open(file_path, 'r') as file:
                data = file.read()
                # Process data to generate report
                return f"Report:\n{data}"
        except FileNotFoundError:
            return "File not found."
        except IOError:
            return "Error reading file."

# Example usage

print(Report.generate_report("reza.text"))
```

Real-world Application Exercises

26. Create a class Ticket for a movie theater with attributes movie_name, seat_number, and price. Provide methods to display ticket details and apply discounts.

```
class Ticket:
    def __init__(self, movie_name, seat_number, price):
        self.movie_name = movie_name
        self.seat_number = seat_number
```

```
        self.price = price

    def display_details(self):
        return f"Movie: {self.movie_name}, Seat: {self.seat_number}, Price: ${self.price:.2f}"

    def apply_discount(self, discount):
        self.price -= self.price * (discount / 100)

# Example usage
ticket = Ticket("Inception", "A10", 12.00)
print(ticket.display_details())
ticket.apply_discount(10)
print(ticket.display_details())
```

27. Create a class ShoppingCart with methods to add, remove, and display items. Each item should be an object of a class Item with attributes name and price.

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

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

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

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

    def display_items(self):
        return [f"{item.name}: ${item.price:.2f}" for item in self.items]

# Example usage
cart = ShoppingCart()
item1 = Item("Apple", 1.00)
item2 = Item("Banana", 0.50)

cart.add_item(item1)
cart.add_item(item2)
print(cart.display_items())
cart.remove_item(item1)
print(cart.display_items())
```

28. Create a class Restaurant with attributes name and menu (a list of Item objects). Provide methods to add and remove items from the menu.

```
class Restaurant:
    def __init__(self, name):
        self.name = name
        self.menu = []

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

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

    def display_menu(self):
        return [f"{item.name}: ${item.price:.2f}" for item in self.menu]

# Example usage
restaurant = Restaurant("The Food Place")
dish1 = Item("Pasta", 12.00)
dish2 = Item("Salad", 8.00)
```

```
restaurant.add_item(dish1)
restaurant.add_item(dish2)
print(restaurant.display_menu())
```

29. Create a class Flight with attributes flight_number, destination, and passengers (a list of Person objects). Provide methods to add and remove passengers.

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

class Flight:
    def __init__(self, flight_number, destination):
        self.flight_number = flight_number
        self.destination = destination
        self.passengers = []

    def add_passenger(self, person):
        self.passengers.append(person)

    def remove_passenger(self, person):
        self.passengers.remove(person)

    def display_passengers(self):
        return [passenger.name for passenger in self.passengers]

# Example usage
flight = Flight("AB123", "New York")
passenger1 = Person("John Doe")
passenger2 = Person("Jane Smith")

flight.add_passenger(passenger1)
flight.add_passenger(passenger2)
print(flight.display_passengers())
flight.remove_passenger(passenger1)
print(flight.display_passengers())
```

30. Create a class Hotel with attributes name and rooms (a list of Room objects). Each Room should have attributes room_number and is_occupied. Provide methods to book and check-out rooms.

```
class Room:
    def __init__(self, room_number):
        self.room_number = room_number
        self.is_occupied = False

class Hotel:
    def __init__(self, name):
        self.name = name
        self.rooms = []

    def add_room(self, room):
        self.rooms.append(room)

    def book_room(self, room_number):
        for room in self.rooms:
            if room.room_number == room_number:
                if not room.is_occupied:
                    room.is_occupied = True
                    return f"Room {room_number} has been booked."
                else:
                    return f"Room {room_number} is already occupied."
        return f"Room {room_number} not found."

    def check_out_room(self, room_number):
        for room in self.rooms:
            if room.room_number == room_number:
                if room.is_occupied:
                    room.is_occupied = False
```

```
                return f"Room {room_number} has been checked out."
            else:
                return f"Room {room_number} is already vacant."
        return f"Room {room_number} not found."

    def display_rooms(self):
        return [(room.room_number, room.is_occupied) for room in self.rooms]

# Example usage
hotel = Hotel("Sunset Inn")
room1 = Room(101)
room2 = Room(102)

hotel.add_room(room1)
hotel.add_room(room2)
print(hotel.display_rooms())
print(hotel.book_room(101))
print(hotel.display_rooms())
print(hotel.check_out_room(101))
print(hotel.display_rooms())
```

GUI Application Exercises

36. Create a class CounterApp that uses tkinter to create a simple counter GUI with increment and decrement buttons.

```
import tkinter as tk

class CounterApp:
    def __init__(self, root):
        self.counter = 0
        self.label = tk.Label(root, text=str(self.counter), font=("Arial", 24))
        self.label.pack()

        self.increment_button = tk.Button(root, text="Increment", command=self.increment)
        self.increment_button.pack(side=tk.LEFT)

        self.decrement_button = tk.Button(root, text="Decrement", command=self.decrement)
        self.decrement_button.pack(side=tk.RIGHT)

    def increment(self):
        self.counter += 1
        self.label.config(text=str(self.counter))

    def decrement(self):
        self.counter -= 1
        self.label.config(text=str(self.counter))

# Example usage
if __name__ == "__main__":
    root = tk.Tk()
    root.title("Counter App")
    app = CounterApp(root)
    root.mainloop()
```

37. Create a class ToDoApp that uses tkinter to create a to-do list GUI where users can add and remove tasks.

```
import tkinter as tk
from tkinter import simpledialog, messagebox

class ToDoApp:
    def __init__(self, root):
        self.root = root
        self.root.title("To-Do List App")

        self.tasks = []

        self.task_listbox = tk.Listbox(root, height=10, width=50)
        self.task_listbox.pack()
```

```
        self.add_button = tk.Button(root, text="Add Task", command=self.add_task)
        self.add_button.pack(side=tk.LEFT)

        self.remove_button = tk.Button(root, text="Remove Task", command=self.remove_task)
        self.remove_button.pack(side=tk.RIGHT)

    def add_task(self):
        task = simpledialog.askstring("Input", "Enter the task:")
        if task:
            self.tasks.append(task)
            self.update_task_listbox()

    def remove_task(self):
        selected_task_index = self.task_listbox.curselection()
        if selected_task_index:
            task = self.tasks.pop(selected_task_index[0])
            self.update_task_listbox()
            messagebox.showinfo("Task Removed", f"Removed task: {task}")

    def update_task_listbox(self):
        self.task_listbox.delete(0, tk.END)
        for task in self.tasks:
            self.task_listbox.insert(tk.END, task)

# Example usage
if __name__ == "__main__":
    root = tk.Tk()
    app = ToDoApp(root)
    root.mainloop()
```

38. Create a class CalculatorApp that uses tkinter to create a simple calculator GUI.import tkinter as tk

```
class CalculatorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Calculator")

        self.expression = ""
        self.result_var = tk.StringVar()

        self.entry = tk.Entry(root, textvariable=self.result_var, font=("Arial", 18), bd=10, insertwidth=4, width=14,
borderwidth=4)
        self.entry.grid(row=0, column=0, columnspan=4)

        buttons = [
            ('7', 1, 0), ('8', 1, 1), ('9', 1, 2), ('/', 1, 3),
            ('4', 2, 0), ('5', 2, 1), ('6', 2, 2), ('*', 2, 3),
            ('1', 3, 0), ('2', 3, 1), ('3', 3, 2), ('-', 3, 3),
            ('0', 4, 0), ('.', 4, 1), ('+', 4, 2), ('=', 4, 3),
            ('C', 5, 0)
        ]

        for (text, row, col) in buttons:
            self.create_button(text, row, col)

    def create_button(self, text, row, col):
        button = tk.Button(self.root, text=text, padx=20, pady=20, font=("Arial", 18), command=lambda:
self.on_button_click(text))
        button.grid(row=row, column=col, sticky="nsew")

    def on_button_click(self, char):
        if char == 'C':
            self.expression = ""
        elif char == '=':
            try:
```

```
                self.expression = str(eval(self.expression))
            except:
                self.expression = "Error"
        else:
            self.expression += str(char)
        self.result_var.set(self.expression)

# Example usage
if __name__ == "__main__":
    root = tk.Tk()
    app = CalculatorApp(root)
    root.mainloop()
```

39. Create a class LoginApp that uses tkinter to create a login form GUI.

```
import tkinter as tk
from tkinter import messagebox

class LoginApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Login Form")

        self.username_label = tk.Label(root, text="Username")
        self.username_label.pack()
        self.username_entry = tk.Entry(root)
        self.username_entry.pack()

        self.password_label = tk.Label(root, text="Password")
        self.password_label.pack()
        self.password_entry = tk.Entry(root, show='*')
        self.password_entry.pack()

        self.login_button = tk.Button(root, text="Login", command=self.login)
        self.login_button.pack()

    def login(self):
        username = self.username_entry.get()
        password = self.password_entry.get()
        if username == "admin" and password == "password":
            messagebox.showinfo("Login Success", "Welcome!")
        else:
            messagebox.showerror("Login Failed", "Invalid credentials")

# Example usage
if __name__ == "__main__":
    root = tk.Tk()
    app = LoginApp(root)
    root.mainloop()
```

40. Create a class WeatherApp that uses tkinter to create a weather information GUI.

```
import tkinter as tk
import requests

class WeatherApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Weather App")

        self.city_label = tk.Label(root, text="City:")
        self.city_label.pack()
        self.city_entry = tk.Entry(root)
        self.city_entry.pack()

        self.get_weather_button = tk.Button(root, text="Get Weather", command=self.get_weather)
        self.get_weather_button.pack()

        self.weather_info_label = tk.Label(root, text="", font=("Arial", 18))
```

```
        self.weather_info_label.pack()

    def get_weather(self):
        city = self.city_entry.get()
        api_key = "your_api_key_here"  # Replace with your actual API key
        url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric"

        try:
            response = requests.get(url)
            weather_data = response.json()
            if weather_data["cod"] == 200:
                temp = weather_data["main"]["temp"]
                desc = weather_data["weather"][0]["description"]
                self.weather_info_label.config(text=f"Temperature: {temp}°C\nDescription: {desc}")
            else:
                self.weather_info_label.config(text="City not found")
        except requests.exceptions.RequestException as e:
            self.weather_info_label.config(text="Error fetching weather data")

# Example usage
if __name__ == "__main__":
    root = tk.Tk()
    app = WeatherApp(root)
    root.mainloop()
```

Basic Class and Object Exercises

1)Create a class called Person with attributes name and age. Create an object of the class and print its attributes.

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

person = Person("Alice", 30)
print(person.name, person.age)
```

2)Add a method called greet to the Person class that prints a greeting message including the person's name.

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

    def greet(self):
        print(f"Hello, my name is {self.name}.")

person = Person("Alice", 30)
person.greet()
```

3)Create a class called Car with attributes make, model, and year. Include a method to print out the car's details.

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def print_details(self):
        print(f"Car details: {self.year} {self.make} {self.model}")

car = Car("Toyota", "Corolla", 2020)
car.print_details()
```

4)Create a class Circle with a method to compute the area. Initialize the class with the radius.

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

```
    def compute_area(self):
        return 3.14159 * self.radius * self.radius

circle = Circle(5)
print(circle.compute_area())
```

5)Create a class Rectangle with methods to compute the area and perimeter. Initialize the class with the length and width.

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def compute_area(self):
        return self.length * self.width

    def compute_perimeter(self):
        return 2 * (self.length + self.width)

rectangle = Rectangle(4, 5)
print(rectangle.compute_area())
print(rectangle.compute_perimeter())
```

Inheritance and Polymorphism Exercises

6)Create a base class Animal with a method speak. Create two derived classes Dog and Cat that override the speak method.

```
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "Woof"

class Cat(Animal):
    def speak(self):
        return "Meow"

dog = Dog()
cat = Cat()
print(dog.speak())
print(cat.speak())
```

7)Create a base class Shape with a method area. Create derived classes Square and Triangle that implement the area method.

```
class Shape:
    def area(self):
        pass

class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side * self.side

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

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

square = Square(4)
triangle = Triangle(3, 6)
print(square.area())
print(triangle.area())
```

8)Create a class Employee with attributes name and salary. Create a derived class Manager with an additional attribute department.

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

class Manager(Employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

manager = Manager("Alice", 80000, "HR")
print(manager.name, manager.salary, manager.department)
```

9)Create a base class Vehicle with a method drive. Create derived classes Bike and Truck that override the drive method.

```
class Vehicle:
    def drive(self):
        pass

class Bike(Vehicle):
    def drive(self):
        return "Bike is driving"

class Truck(Vehicle):
    def drive(self):
        return "Truck is driving"

bike = Bike()
truck = Truck()
print(bike.drive())
print(truck.drive())
```

10)Create a base class Bird with a method fly.
Create derived classes Eagle and Penguin. Override the fly method in Penguin to indicate that penguins cannot fly.

```
class Bird:
    def fly(self):
        pass

class Eagle(Bird):
    def fly(self):
        return "Eagle is flying"

class Penguin(Bird):
    def fly(self):
        return "Penguins cannot fly"

eagle = Eagle()
penguin = Penguin()
print(eagle.fly())
print(penguin.fly())
```

EncapsulationandAbstractionExercises

11)Create a class Account with private attributes balance. Provide public methods to deposit and withdraw money.

```
class Account:
    def __init__(self, balance):
        self.__balance = balance

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

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
```

```
    def get_balance(self):
        return self.__balance

account = Account(1000)
account.deposit(500)
account.withdraw(200)
print(account.get_balance())
```

12)Create a class Book with private attributes title, author, and pages. Provide public methods to get and set these attributes.

```
class Book:
    def __init__(self, title, author, pages):
        self.__title = title
        self.__author = author
        self.__pages = pages

    def get_title(self):
        return self.__title

    def get_author(self):
        return self.__author

    def get_pages(self):
        return self.__pages

    def set_title(self, title):
        self.__title = title

    def set_author(self, author):
        self.__author = author

    def set_pages(self, pages):
        self.__pages = pages

book = Book("1984", "George Orwell", 328)
print(book.get_title())
print(book.get_author())
print(book.get_pages())
```

13)Create a class Laptop with private attributes brand, model, and price. Provide a method to apply a discount and a method to display laptop details.

```
class Laptop:
    def __init__(self, brand, model, price):
        self.__brand = brand
        self.__model = model
        self.__price = price

    def apply_discount(self, discount):
        self.__price -= discount

    def display_details(self):
        return f"Laptop: {self.__brand} {self.__model}, Price: {self.__price}"

laptop = Laptop("Dell", "XPS 15", 1500)
laptop.apply_discount(200)
print(laptop.display_details())
```

14)Create a class BankAccount with private attributes account_number and balance. Provide methods to deposit, withdraw, and check the balance.

```
class BankAccount:
    def __init__(self, account_number, balance):
        self.__account_number = account_number
        self.__balance = balance

    def deposit(self, amount):
```

```
        self.__balance += amount

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount

    def check_balance(self):
        return self.__balance

bank_account = BankAccount("123456789", 1000)
bank_account.deposit(500)
bank_account.withdraw(300)
print(bank_account.check_balance())
```

15)Create a class Student with private attributes name, grade, and age. Provide methods to get and set these attributes and a method to display the student's details.

```
class Student:
    def __init__(self, name, grade, age):
        self.__name = name
        self.__grade = grade
        self.__age = age

    def get_name(self):
        return self.__name

    def get_grade(self):
        return self.__grade

    def get_age(self):
        return self.__age

    def set_name(self, name):
        self.__name = name

    def set_grade(self, grade):
        self.__grade = grade

    def set_age(self, age):
        self.__age = age

    def display_details(self):
        return f"Name: {self.__name}, Grade: {self.__grade}, Age: {self.__age}"

student = Student("John", "A", 20)
print(student.display_details())
```

Class Relationships and Advanced Concepts Exercises

16. Create a class Library with attributes name and books (a list of Book objects). Provide methods to add and remove books.

```
class Book:
    def __init__(self,title,author):
        self.title = title
        self.author = author
class Library:
    def __init__(self,name):
        self.name = name
        self.books = []
    def add_books(self,book):
        self.books.append(book)
    def remove_books(self,book):
        self.books.remove(book)

library = Library("My Library")
book1 = Book("kabul","Alisina")
book2 = Book("Mmountain","Ahmad")
```

```
library.add_books(book1)
library.add_books(book2)
print([ book.title for book in library.books])
library.remove_books(book1)
print([ book.title for book in library.books])
```

17. Create a class School with attributes name and students (a list of Student objects). Provide methods to add and remove students.

```
class Student:
    def __init__(self,name,grade):
        self.name = name
        self.grade = grade
class School:
    def __init__(self,name):
        self.name = name
        self.students = []
    def add_students(self,student):
        self.students.append(student)
    def remove_students(self,student):
        self.students.remove(student)

school = School("Moradi")
student1 = Student("alisina",12)
student2= Student("Ahmad",10)

school.add_students(student1)
school.add_students(student2)
print([ student.name for student in school.students])
school.remove_students(student1)
print([ student.name for student in school.students])
```

18. Create a class Team with attributes name and members (a list of Person objects). Provide methods to add and remove members.

```
class Person:
    def __init__(self,name,grade):
        self.name = name
        self.grade = grade
class Team:
    def __init__(self,name):
        self.name = name
        self.persons = []
    def add_persons(self,person):
        self.persons.append(person)
    def remove_persons(self,person):
        self.persons.remove(person)

team = Team("Moradi")
person1 = Person("alisina",12)
person2= Person("Ahmad",10)

team.add_persons(person1)
team.add_persons(person2)
print([ person.name for person in team.persons])
team.remove_persons(person1)
print([ person.name for person in team.persons])
```

19. Create a class Company with attributes name and employees (a list of Employee objects). Provide methods to add and remove employees.

```
class Emploee:
    def __init__(self,name,grade):
        self.name = name
        self.grade = grade
class Company:
    def __init__(self,name):
        self.name = name
        self.emploees = []
```

```
    def add_emploees(self,emploee):
        self.emploees.append(emploee)
    def remove_emploees(self,emploee):
        self.emploees.remove(emploee)

company = Company("Moradi")
emploee1 = Emploee("alisina",12)
emploee2= Emploee("Ahmad",10)

company.add_emploees(emploee1)
company.add_emploees(emploee2)
print([ emploee.name for emploee in company.emploees])
company.remove_emploees(emploee1)
print([ emploee.name for emploee in company.emploees])
```

20. Create a class Zoo with attributes name and animals (a list of Animal objects). Provide methods to add and remove animals.

```
class Animal:
    def __init__(self,name,grade):
        self.name = name
        self.grade = grade
class Zoo:
    def __init__(self,name):
        self.name = name
        self.animals = []
    def add_animals(self,animal):
        self.animals.append(animal)
    def remove_animals(self,animal):
        self.animals.remove(animal)

zoo = Zoo("Nora")
animal1 = Animal("zebra",12)
animal2= Animal("lion",10)

zoo.add_animals(animal1)
zoo.add_animals(animal2)
print([animal.name for animal in zoo.animals])
zoo.remove_animals(animal1)
print([animal.name for animal in zoo.animals])
```

File Handling and Exceptions Exercises

21. Create a class FileManager with methods to read from and write to a file.

```
class FileManager:
    @staticmethod
    def write_file(content):
        with open("mahmod.text", 'w') as file:
            file.write(content)

    @staticmethod
    def read_file(file_path):
        with open(file_path, 'r') as file:
            return file.read()

# Example usage
#file_manager = FileManager()
FileManager.write_file('Hello, i am good  and World!')
print(FileManager.read_file('mahmod.text'))
```

22. Create a class Log with methods to write error messages to a log file.

```
import datetime
class Log:
    @staticmethod
    def write_file(massage):
        with open("reza.text","a") as file:
            file.write(f"{datetime.datetime.now()}-ERROR:{massage}\n")
```

```
    @staticmethod
    def read_file(file_path):
        with open(file_path,"r") as file:
            return file.read()

Log.write_file("An error ocured ok")
print(Log.read_file("reza.text"))
```

23. Create a class Config that reads configuration settings from a file and provides methods to access these settings.

```
class Config:
    def __init__(self, filename):
        self.settings = {}
        self.load_config(filename)

    def load_config(self, filename):
        with open(filename, 'r') as file:
            for line in file:
                key, value = line.strip().split("=")
                self.settings[key.strip()] = value.strip()

    def get(self, key, default=None):
        return self.settings.get(key, default)

config = Config('config.txt')
print(config.get('name'))    # خروجی: alisina
print(config.get('grade'))   # خروجی: 12
print(config.get('address')) # خروجی: barchi
```

24. Create a class Database that connects to a database and provides methods to execute queries. Handle exceptions if the connection fails.

```
import sqlite3
class Database:
    def __init__(self, db_name):
        self.db_name = db_name
        self.connection = None

    def connect(self):
        try:
            self.connection = sqlite3.connect(self.db_name)
        except sqlite3.Error as e:
            print(f"Connection failed: {e}")

    def execute_query(self, query):
        if self.connection:
            try:
                cursor = self.connection.cursor()
                cursor.execute(query)
                self.connection.commit()
            except sqlite3.Error as e:
                print(f"Query failed: {e}")

# Example usage
db = Database('example.db')
db.connect()
db.execute_query('CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)')
```

25. Create a class Report that generates a report from data in a file. Provide methods to handle exceptions if the file does not exist or cannot be read.

```
class Report:
    @staticmethod
    def generate_report(file_path):
        try:
            with open(file_path, 'r') as file:
```

```
            data = file.read()
            # Process data to generate report
            return f"Report:\n{data}"
        except FileNotFoundError:
            return "File not found."
        except IOError:
            return "Error reading file."

# Example usage

print(Report.generate_report("reza.text"))
```

Real-world Application Exercises

26. Create a class Ticket for a movie theater with attributes movie_name, seat_number, and price. Provide methods to display ticket details and apply discounts.

```
class Ticket:
    def __init__(self, movie_name, seat_number, price):
        self.movie_name = movie_name
        self.seat_number = seat_number
        self.price = price

    def display_details(self):
        return f"Movie: {self.movie_name}, Seat: {self.seat_number}, Price: ${self.price:.2f}"

    def apply_discount(self, discount):
        self.price -= self.price * (discount / 100)

# Example usage
ticket = Ticket("Inception", "A10", 12.00)
print(ticket.display_details())
ticket.apply_discount(10)
print(ticket.display_details())
```

27. Create a class ShoppingCart with methods to add, remove, and display items. Each item should be an object of a class Item with attributes name and price.

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

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

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

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

    def display_items(self):
        return [f"{item.name}: ${item.price:.2f}" for item in self.items]

# Example usage
cart = ShoppingCart()
item1 = Item("Apple", 1.00)
item2 = Item("Banana", 0.50)

cart.add_item(item1)
cart.add_item(item2)
print(cart.display_items())
cart.remove_item(item1)
print(cart.display_items())
```

28. Create a class Restaurant with attributes name and menu (a list of Item objects). Provide methods to add and remove

items from the menu.

```
class Restaurant:
    def __init__(self, name):
        self.name = name
        self.menu = []

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

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

    def display_menu(self):
        return [f"{item.name}: ${item.price:.2f}" for item in self.menu]

# Example usage
restaurant = Restaurant("The Food Place")
dish1 = Item("Pasta", 12.00)
dish2 = Item("Salad", 8.00)

restaurant.add_item(dish1)
restaurant.add_item(dish2)
print(restaurant.display_menu())
```

29. Create a class Flight with attributes flight_number, destination, and passengers (a list of Person objects). Provide methods to add and remove passengers.

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

class Flight:
    def __init__(self, flight_number, destination):
        self.flight_number = flight_number
        self.destination = destination
        self.passengers = []

    def add_passenger(self, person):
        self.passengers.append(person)

    def remove_passenger(self, person):
        self.passengers.remove(person)

    def display_passengers(self):
        return [passenger.name for passenger in self.passengers]

# Example usage
flight = Flight("AB123", "New York")
passenger1 = Person("John Doe")
passenger2 = Person("Jane Smith")

flight.add_passenger(passenger1)
flight.add_passenger(passenger2)
print(flight.display_passengers())
flight.remove_passenger(passenger1)
print(flight.display_passengers())
```

30. Create a class Hotel with attributes name and rooms (a list of Room objects). Each Room should have attributes room_number and is_occupied. Provide methods to book and check-out rooms.

```
class Room:
    def __init__(self, room_number):
        self.room_number = room_number
        self.is_occupied = False

class Hotel:
    def __init__(self, name):
        self.name = name
```

```
        self.rooms = []

    def add_room(self, room):
        self.rooms.append(room)

    def book_room(self, room_number):
        for room in self.rooms:
            if room.room_number == room_number:
                if not room.is_occupied:
                    room.is_occupied = True
                    return f"Room {room_number} has been booked."
                else:
                    return f"Room {room_number} is already occupied."
        return f"Room {room_number} not found."

    def check_out_room(self, room_number):
        for room in self.rooms:
            if room.room_number == room_number:
                if room.is_occupied:
                    room.is_occupied = False
                    return f"Room {room_number} has been checked out."
                else:
                    return f"Room {room_number} is already vacant."
        return f"Room {room_number} not found."

    def display_rooms(self):
        return [(room.room_number, room.is_occupied) for room in self.rooms]

# Example usage
hotel = Hotel("Sunset Inn")
room1 = Room(101)
room2 = Room(102)

hotel.add_room(room1)
hotel.add_room(room2)
print(hotel.display_rooms())
print(hotel.book_room(101))
print(hotel.display_rooms())
print(hotel.check_out_room(101))
print(hotel.display_rooms())
```

GUI Application Exercises

36. Create a class CounterApp that uses tkinter to create a simple counter GUI with increment and decrement buttons.

```
import tkinter as tk

class CounterApp:
    def __init__(self, root):
        self.counter = 0
        self.label = tk.Label(root, text=str(self.counter), font=("Arial", 24))
        self.label.pack()

        self.increment_button = tk.Button(root, text="Increment", command=self.increment)
        self.increment_button.pack(side=tk.LEFT)

        self.decrement_button = tk.Button(root, text="Decrement", command=self.decrement)
        self.decrement_button.pack(side=tk.RIGHT)

    def increment(self):
        self.counter += 1
        self.label.config(text=str(self.counter))

    def decrement(self):
        self.counter -= 1
        self.label.config(text=str(self.counter))
```

```
# Example usage
if __name__ == "__main__":
    root = tk.Tk()
    root.title("Counter App")
    app = CounterApp(root)
    root.mainloop()
```

37. Create a class ToDoApp that uses tkinter to create a to-do list GUI where users can add and remove tasks.import tkinter as tk

```
from tkinter import simpledialog, messagebox

class ToDoApp:
    def __init__(self, root):
        self.root = root
        self.root.title("To-Do List App")

        self.tasks = []

        self.task_listbox = tk.Listbox(root, height=10, width=50)
        self.task_listbox.pack()

        self.add_button = tk.Button(root, text="Add Task", command=self.add_task)
        self.add_button.pack(side=tk.LEFT)

        self.remove_button = tk.Button(root, text="Remove Task", command=self.remove_task)
        self.remove_button.pack(side=tk.RIGHT)

    def add_task(self):
        task = simpledialog.askstring("Input", "Enter the task:")
        if task:
            self.tasks.append(task)
            self.update_task_listbox()

    def remove_task(self):
        selected_task_index = self.task_listbox.curselection()
        if selected_task_index:
            task = self.tasks.pop(selected_task_index[0])
            self.update_task_listbox()
            messagebox.showinfo("Task Removed", f"Removed task: {task}")

    def update_task_listbox(self):
        self.task_listbox.delete(0, tk.END)
        for task in self.tasks:
            self.task_listbox.insert(tk.END, task)

# Example usage
if __name__ == "__main__":
    root = tk.Tk()
    app = ToDoApp(root)
    root.mainloop()
```

38. Create a class CalculatorApp that uses tkinter to create a simple calculator GUI.import tkinter as tk

```
class CalculatorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Calculator")

        self.expression = ""
        self.result_var = tk.StringVar()

        self.entry = tk.Entry(root, textvariable=self.result_var, font=("Arial", 18), bd=10, insertwidth=4, width=14,
borderwidth=4)
        self.entry.grid(row=0, column=0, columnspan=4)

        buttons = [
```

```
            ('7', 1, 0), ('8', 1, 1), ('9', 1, 2), ('/', 1, 3),
            ('4', 2, 0), ('5', 2, 1), ('6', 2, 2), ('*', 2, 3),
            ('1', 3, 0), ('2', 3, 1), ('3', 3, 2), ('-', 3, 3),
            ('0', 4, 0), ('.', 4, 1), ('+', 4, 2), ('=', 4, 3),
            ('C', 5, 0)
        ]

        for (text, row, col) in buttons:
            self.create_button(text, row, col)

    def create_button(self, text, row, col):
        button = tk.Button(self.root, text=text, padx=20, pady=20, font=("Arial", 18), command=lambda:
self.on_button_click(text))
        button.grid(row=row, column=col, sticky="nsew")

    def on_button_click(self, char):
        if char == 'C':
            self.expression = ""
        elif char == '=':
            try:
                self.expression = str(eval(self.expression))
            except:
                self.expression = "Error"
        else:
            self.expression += str(char)
        self.result_var.set(self.expression)

# Example usage
if __name__ == "__main__":
    root = tk.Tk()
    app = CalculatorApp(root)
    root.mainloop()
```

39. Create a class LoginApp that uses tkinter to create a login form GUI.

```
import tkinter as tk
from tkinter import messagebox

class LoginApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Login Form")

        self.username_label = tk.Label(root, text="Username")
        self.username_label.pack()
        self.username_entry = tk.Entry(root)
        self.username_entry.pack()

        self.password_label = tk.Label(root, text="Password")
        self.password_label.pack()
        self.password_entry = tk.Entry(root, show='*')
        self.password_entry.pack()

        self.login_button = tk.Button(root, text="Login", command=self.login)
        self.login_button.pack()

    def login(self):
        username = self.username_entry.get()
        password = self.password_entry.get()
        if username == "admin" and password == "password":
            messagebox.showinfo("Login Success", "Welcome!")
        else:
            messagebox.showerror("Login Failed", "Invalid credentials")

# Example usage
if __name__ == "__main__":
    root = tk.Tk()
```

```
    app = LoginApp(root)
    root.mainloop()
```

40. Create a class WeatherApp that uses tkinter to create a weather information GUI.import tkinter as tk
import requests

```
class WeatherApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Weather App")

        self.city_label = tk.Label(root, text="City:")
        self.city_label.pack()
        self.city_entry = tk.Entry(root)
        self.city_entry.pack()

        self.get_weather_button = tk.Button(root, text="Get Weather", command=self.get_weather)
        self.get_weather_button.pack()

        self.weather_info_label = tk.Label(root, text="", font=("Arial", 18))
        self.weather_info_label.pack()

    def get_weather(self):
        city = self.city_entry.get()
        api_key = "your_api_key_here"  # Replace with your actual API key
        url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric"

        try:
            response = requests.get(url)
            weather_data = response.json()
            if weather_data["cod"] == 200:
                temp = weather_data["main"]["temp"]
                desc = weather_data["weather"][0]["description"]
                self.weather_info_label.config(text=f"Temperature: {temp}°C\nDescription: {desc}")
            else:
                self.weather_info_label.config(text="City not found")
        except requests.exceptions.RequestException as e:
            self.weather_info_label.config(text="Error fetching weather data")

# Example usage
if __name__ == "__main__":
    root = tk.Tk()
    app = WeatherApp(root)
    root.mainloop()
```

BUILTING FUNCTIONS

```
# Example of abs()
absolute_value = abs(-20)
print('Absolute value of -20 is:', absolute_value)

# Example of all()
all_true = all([True, True, True])
print('All elements are True:', all_true)

# Example of any()
any_true = any([False, False, True])
print('Any element is True:', any_true)

# Example of ascii()
ascii_representation = ascii('hello')
print('ASCII representation of "hello":', ascii_representation)
```

```
# Example of bin()
binary_representation = bin(12)
print('Binary representation of 12:', binary_representation)

# Example of bool()
boolean_value = bool(0)
print('Boolean value of 0:', boolean_value)

# Example of bytearray()
byte_array = bytearray(3)
print('Bytearray of size 3:', byte_array)

# Example of bytes()
bytes_object = bytes(4)
print('Bytes object of size 4:', bytes_object)

# Using the int() function to convert a string to an integer
number = int("42")
print(f"int('42') -> {number}")  # Output: 42

# Using isinstance() to check if an object is an instance of a class
value = 42
print(f"isinstance(42, int) -> {isinstance(value, int)}")  # Output: True

# Using issubclass() to check if a class is a subclass of another class
class Animal:
    pass

class Dog(Animal):
    pass

print(f"issubclass(Dog, Animal) -> {issubclass(Dog, Animal)}")  # Output: True

# Using iter() to create an iterator
my_list = [1, 2, 3]
iterator = iter(my_list)
print(f"next(iter([1, 2, 3])) -> {next(iterator)}")  # Output: 1

# Using len() to get the length of a string
message = "Hello, World!"
print(f"len('Hello, World!') -> {len(message)}")  # Output: 13

# Using list() to create a list from a string
characters = list("abc")
print(f"list('abc') -> {characters}")  # Output: ['a', 'b', 'c']

# Using locals() to get a dictionary of local variables
def sample_function():
    test_var = "example"
    return locals()

local_variables = sample_function()
print(f"locals() in sample_function() -> {local_variables}")  # Output: {'test_var': 'example'}

# Using map() to apply a function to a list
def square(x):
    return x * x

numbers = [1, 2, 3, 4]
squared_numbers = list(map(square, numbers))
```

```
print(f"map(square, [1, 2, 3, 4]) -> {squared_numbers}")  # Output: [1, 4, 9, 16]

# Printing a welcome message
print("Hello, World!")

# Using a property in a Python class to manage attribute access
class Circle:
    def __init__(self, radius):
        self._radius = radius

    @property
    def radius(self):
        "The radius property gets the value of the radius"
        return self._radius

    @radius.setter
    def radius(self, value):
        "The radius property sets the value of the radius"
        if value < 0:
            raise ValueError("Radius cannot be negative")
        self._radius = value

# Creating a circle object and interacting with its radius
circle = Circle(5)
print(f"Initial circle radius: {circle.radius}")
circle.radius = 10
print(f"Updated circle radius: {circle.radius}")

# Iterating over a range of numbers
print("Numbers from 0 to 2:")
for i in range(3):
    print(i)

# Displaying the string representation of an object
x = [1, 2, 3]
print(f"String representation of the list: {repr(x)}")

# Reversing a list and printing it
my_list = [1, 2, 3]
reversed_list = list(reversed(my_list))
print(f"List in reverse order: {reversed_list}")

# Rounding a floating-point number
number = 3.14159
rounded_number = round(number, 2)
print(f"Rounded number: {rounded_number}")

# Creating a set to demonstrate unique elements
my_set = set([1, 2, 2, 3, 4])
print(f"Unique elements from the list: {my_set}")

# Using setattr to dynamically add attributes to an object
class Person:
    def __init__(self, name):
        self.name = name

person = Person("John")
setattr(person, "age", 30)
print(f"Person's name: {person.name}, age: {person.age}")
```

Tower Hanoi

```
def tower_of_hanoi(n, source, target, auxiliary):
    if n == 1:
        print(f"Move disk 1 from {source} to {target}")
        return
    tower_of_hanoi(n - 1, source, auxiliary, target)
    print(f"Move disk {n} from {source} to {target}")
    tower_of_hanoi(n - 1, auxiliary, target, source)

# Number of disks
n = 3
tower_of_hanoi(n, 'A', 'C', 'B')
```

Fibonacci Sequences

```
def fibonacci_recursive(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci_recursive(n-1) + fibonacci_recursive(n-2)

#tast of the function
for i in range(10):
    print(f"Fibonacci({i}) = {fibonacci_recursive(i)}")
```