Name:- Ali Asghar

Father's Name:- Mohammad Jawad

Last Name:- Ali Zada

Faculty:- Computer Science

Department:- IT

#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
ahmad = Person("Ahmad", 18)
print("name:", ahmad.name, '\n', "age:",
ahmad.age)
```

#2. Creat (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):
        self.name = name
    def greeting(self):
        return "Hello"
ahmad = Person("Ahmad")
print(f"{ahmad.greeting()} {ahmad.name}.
```

First

Homework.

```
        self.radius = radius
    def area(self):
        x = 3.14 * ((self.radius) ** 2)
        print("Area:", x)
cir = Circle(2)
cir.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 area():
        a = (self.length * self.width)
        print("Area:", a)
    def perimeter(self):
        p = (2 * self.length) + (2 * self.width)
        print("perimeter:", p)
rect = Rectangle(2, 4)
rect.area()
```

How are you doing?")

#3. Creat 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 show_details(self):
        print("Make:", self.make)
        print("Model:", self.model)
        print("Year:", self.year)

lamborghini = Car("Lamborghini", "Miura P400", 2018)
lamborghini.show_details()
```

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

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

the area method.

```
class Shape:
    def area(self):
        a = self.length * self.width
        print("Area:", a)
class Square(Shape):
    def __init__(self, base, hieght):
        self.length = length
        self.width = width
class Triangle(Shape):
    def __init__(self, base, hiegth):
        self.base = base
        self.hiegth = hiegth
    def area(self):
        a = 1/2 * self.base * self.hieght
        print("Area:", a)
shap1 = Square(2, 4)
shap1.area()
shap2 = Triangle(2, 4)
shap2.area()
```

#8. Create a class Employee with attributes name and salary. Create.

rect.perimeter()

#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):
        print("Animal sound")
class Dog(Animal):
    def speak(self):
        print("Hoop Hoop!")
class Cat(Animal):
    def speak(self):
        print("Meow...!")
cat = Cat()
dog = Dog()
cat.speak()
dog.speak()
```

#7. Create a base class shape with a method area. Create derived classes Square and Triangle that implement.

```
    def derive(self):
        print("Driving the Bike")
class Truck(Vehicle):
    def derive(self):
        print("Driving the truck")
vehicle1 = Bike()
vehicle2 = Truck()
vehicle1.derive()
vehicle2.derive()
```

#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 penguin cannot fly.

```
class Bird:
    def fly(self):
        print("Birds can fly")
class Eagle(Bird):
    def fly(self):
        print("Eagles can fly!")
class Penguins(Bird):
    def fly(self):
```

a derived class Manager with an additional attributes 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.deparment = department
employee = Manager("Ahmad", 4000, "IT")
print("Name", employee.name)
print("Salary:", employee.salary)
print("Department:", employee.department)
```

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

```
class Vehicle:
    def derive(self):
        print("Driving the vehicle")
class Bike(Vehicle):
```

amount, "withdraw is", self.__balance)
        else:
            print("INSUFFICIENT AMOUNT")
my_money = Account(5000)
my_money.deposite()
my_money.withdraw()

#12. Create a class Book with private attributes title, author, and papes. 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 set_title(self):
        self.__title = title
    def get_title(self):
        return self.__title
    def set_author(self):
        self.__author = author
    def get_author(self):

        print("Penguins cannot fly.")
bird = Penguin()
bird.fly()

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

class Account:
    def __init__(self, balance):
        self.__balance = balance
    def deposit(self):
        amount = int(input("Please, enter your money amount! \n Amount: "))
        self.__balance += amount
        print("Your balance after", amount, "deposite is", self.__balance)
    def withdraw(self):
        amount = int(input("Please, enter your money amount! \n Amount: "))
        if amount < self.__balance:
            self.__balance = amount
            print("Your balance after",

```
is", self.__price)
    def display_details(self):
        print("Brand:", self.__brand)
        print("Model:", self.__model)
        print("Price:", self.__price)
laptop = Laptop("DELL", "7410", 36000)
laptop.discount(4000)
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():
        amount = int(input("Please, enter your deposit amount!\n Amount:"))
        self.__balance += amount
        print("Your balance after", amount, "deposit is", self.__balance)
```

```
        return self.__author
    def set_pages(self):
        self.__pages = pages
    def get_pages(self):
        return self.__author
harry_potter = Book("Harry Potter", "J.K", 300)
print("Title:", harry_potter.get_title())
print("Author:", harry_potter.get_author())
print("Pages:", harry_potter.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 discount(self, discount_amount):
        self.__price -= discount_amount
        print("The price after disaount
```

```
def __init__(self, name, grade, age):
    self.__name = name
    self.__grade = grade
    self.__age = age
def set_name(self):
    self.__name = name
def get_name(self):
    return self.__name
def set_grade(self):
    self.__grade = grade
def get_grade(self):
    return self.grade
def set_age(self):
    self.__age = age
def get_age(self):
    return self.__age
def details(self):
    print("Name:", self.__name)
    print("Grade:", self.__grade)
    print("Age:", self.__age)
Ahmad = Student("Ahmad", 8, 14)
print("Name:", Ahmad.__name)
print("Grade:", Ahmad.__grade)
```

```
def withdraw(self):
    amount = int(input("Please, enter your withdraw amount! \n Amount:"))
    if amount < self.__balance:
        self.__balance -= amount
        print("Your balance after", amount, "withdraw is", self.__balance)
    else:
        print("Insufficient amount")
def balance_statue(self):
    print("Your Balance is", self.__balance)

ali_account = BankAccount("983425", 60000)
ali_account.deposit()
ali_account.withdraw
ali_account.balance_statue()
```

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

```
class Student:
```

```
library = Library("Ketabistan", ["Atomic Habbits", "Never GIVE UP!"])
library.add("Alphabet")
library.remove("Atomic Habbits")
```

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

```
class School:
    def __init__(self, name, students):
        self.name = name
        self.students = students
    def add(self, students):
        self.students.append(student)
        print("Our recently updated student's list is", self.students)
    def remove(self, student):
        if student in self.students:
            self.students.remove(student)
            print("Our recently updated student's list is", self.students)
        else:
```

```
            print("Age:", Ahmad.age)
Ahmad.cliteaks()
```

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

```
class Library:
    def __init__(self, name, books):
        self.name = name
        self.books = books
    def add(self, item):
        self.books.append(item)
        print("Our recently updated book archieve is", self.books)
    def remove(self, item):
        if item in self.books:
            self.books.remove(item)
            print("Our recently updated book archieve is", self.books)
        else:
            print("Book does not exist!")
```

```
            self.members.remove(person)
            print("Our recently updated
member's list is", self.members)
        else:
            print(person, "has not joined")
team = Team("Afghan Boys", ["Ahmad",
"Ali", "Ramin"]
team.add("Noorzai")
team.remove("Ahmad")
```

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

```
class Company:
    def __init__(self, name, employees):
        self.name = name
        self.employees = employees
    def add(self, employee):
        self.employees.append(employee)
        print("Our recently updated
employee's list is", self.employees)
    def remove(self, employee):
        print(student, "has not
registered")
abdulRahimShaheed = School("Abdul
Rahim Shaheed High School", ["Ahmad",
"Ali", "Ramin"]
abdulRahimShaheed.add("Noorzai")
abdulRahimShaheed.remove("Ahmad")
```

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

```
class Team:
    def __init__(self, name, members):
        self.name = name
        self.members = members
    def add(self, person):
        self.members.append(person)
        print("Our recently updated
members list is", self.members)
    def remove(self, person):
        self.members
        if person in self.members:
```

```
animals, "animals in our zoo")
    def remove(self, animal):
        if animal in self.animals:
            self.animals.remove(animal)
            print("We have these", self.
animals, "animals in our zoo")
        else:
            print(animal, "did not
bring here!")
zoo = Zoo("Kabul Zoo", ["Lions", "Tigers",
"Rabbits"])
zoo.add("Bear")
zoo.remove("Lions")
```

Real-world Aplication exercises-

#21- 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
```

```
        if employee in self.employees:
            self.employees.remove(employee)
            print("Our recently updated
employee's list is", self.employees)
        else:
            print(employee, "has not
joined")
company = Company("SamSung", ["Ahmad", "Ali",
"Ramin"])
company.add("Noorzai")
company.remove("Ahmad")
```

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

```
class Zoo:
    def __init__(self, name, animals):
        self.name = name
        self.animals = animals
    def add(self, animals):
        self.animals.append(animal)
        print("We have these", self.
```

```
    def __repr__(self):
        return f"{self.name}: ${self.price}"
class ShoppingCart:
    def __init__(self):
        self.items = []
    def add(self, item):
        if isinstance(item, Item):
            self.items.append(item)
            print(item, "added!")
        else:
            print("No item to add!")
    def remove(self, item):
        if item in self.items:
            self.items.remove(item)
            print(item, "removed!")
        else:
            print("Invalid item!")
    def display(self):
        if not self.items:
            print("Shopping cart is empty!")
        else:
            print("Shopping cart:", self.items)
item1 = Item("Apple", 120)
```

```
    def details(self):
        print("Movie Name:", self.movie_name)
        print("Seat Number:", self.seat_number)
        print("Price:", self.price)
    def discount(self):
        discount = 50
        self.price -= discount
        print("The final price for this
movie is", self.price)
ticket = Ticket("Titanic", 24, 250)
ticket.details()
ticket.discount()
```

#22- 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
    def __str__(self):
        return f"{self.name}
```

```
        if isinstance(item, Item):
            self.menu.append(item)
            print(item, "added")
        else:
            print("No Item To Add")
    def remove(self, item):
        if item in self.menu:
            self.menu.remove(item)
            print(item, "removed")
        else:
            print("item does not exist")
it1 = Item("kabab")
it2 = Item("Qabli")
it3 = Item("Manto")
re = Restaurant("Zarafat")
re.add(it1)
re.add(it2)
re.add(it3)
re.remove(it3)
```

#24 Create a class Flight with attributes flight_number, destination, and passengers (a list of person objects). Provide methods


```
item2 = Item("Banana", 60)
item3 = Item("Peach", 200)
cart = Shopping(cart())
card.add(item1)
cart.remove(item2)
cart.add(item2)
cart.add(item3)
cart.display()
```

#23. 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 Item:
    def __init__(self, name):
        self.name = name
    def __repr__(self):
        return f"{self.name}"
class Restaurant:
    def __init__(self, name):
        self.name = name
        self.menu = []
    def add(self, item):
```

f1.add_p(p1)
f1.add_p(p2)
f1.add_p(p3)
f1.remove(p1)

#25 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 checkout rooms.

```
class Room:
    def __init__(self, room_number):
        self.room_number = room_number
        self.is_occupied = False
    def __repr__(self):
        return f"Room{self.room_number}: {status}"
class Hotel:
    def __init__(self, name):
        self.name = name
        self.rooms = []
    def add(self, room):
        if isinstance(room, Room):
            self.rooms.append(room)
```

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_p(self, person):
        if isinstance(person, Person):
            self.passengers.append(person)
            print(Person, "recently added to flight")
    def remove(self, person):
        if person in self.passengers:
            self.passengers.remove(person)
            print(Person, "recently removed")
        else:
            print("No edited")
p1 = Person("Ahmad")
p2 = Person("Ali")
p3 = Person("Hassan")
f1 = Flight("PS342", "Paris")
```

```
    def display(self):
        print("Hotel:", self.name)
        for room in self.rooms:
            print(room)
room1 = Room(45)
hotel = Hotel("Afghan Plaza")
hotel.add(room1)
hotel.display()
hotel.book(45)
hotel.display()
hotel.check_out(45)
hotel.display()
```

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

```
class FileManager
    def __init__(self, file_name):
        self.file_name = file_name
    def write_to_file(self, text):
        with open(self.file_name, 'w') as file:
            file.write(text)
        print(f"Data written to {self.file_name}")
```

```
            print("Room", room.room_number, "added")
        else:
            print("Not added")
    def book(self, room_number):
        for room in self.rooms:
            if room.room_number == room_number:
                if not room.is_occupied:
                    room.is_occupied = True
                    print("Room", room_number, "has been booked.")
                else:
                    print("Room", room_number, "is already occupied.")
    def check_out(self, room_number):
        for room in self.rooms:
            if room.room_number == room_number:
                if room.is_occupied:
                    room.is_occupied = False
                    print("Room", room_number, "has been check out.")
                else:
                    print("Room", room_number, "is not already occupied")
```

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

```
import os
from datatime import datatime
class Log:
    def __init__(self, log_file):
        self.log_file = log_file
        if not os.path.exists(self.log_file):
            open(self.log_file, "w").close()
    def log_error(self, message):
        timestamp = f"[{timestamp}] ERROR: {message}\n"
        log_message = f"[{timestamp}] ERROR: {message}\n"
        with open(self.log_file, 'a') as file:
            file.write(log_message)
        print(f"Logged error: {log_message.strip()}")
    def log_info(self, message):
        timestamp = datatime.now().strftime('%Y-%m-%d %H:%M:%S')
        log_message = f"[{timestamp}] Info: {message}\n"
    def append_to_file(self, text):
        with open(self.file_nam, 'a') as file:
            file.write(text)
        print("Data appended to {self.file name}")
    def read_from_file(self):
        try:
            with open(self.file_name, 'r') as file:
                content = file.read()
            return content
        except FileNotFoundError:
            print(f"The file {self.file_name} does not exist.")
            return None
if __name__ == "__main":
    file_manager = FileManager('example.txt')
    file_manage.write_to_file('Hello, world!\nthis is a test.')
    file_manager.append('\nAppending some more text.')
    content = file_manager.read_from_file()
    print("File content:")
    print(content)
```

```
Import json
class Config:
    def __init__(self, filename = 'config.json'):
        self.filename= filename
        self. settings =self. load_config()
    def load_config(self):
        try:
            with open (self.filename, 'r') as file:
                return json.load(file)
        except (FileNotFoundError, json.JSON
DecodeError) as e:
                print(f"Error loading config {e}")
                return{}
    def get(self, key, default = None):
        return self.settings. get(key, default)
if __name__ == "__main__":
    coting= Config()
    db_host= config. get('db_host', 'localhost')
    db_port= config.get('db_host', 5432)
    print (f"database Host: {db_host}")
    print(f"Database Port: {db_port}")
```

#29_ Create a class data base ~~Host~~:

```
    with open (self.log_file, 'a') as file:
        file. write( log_message)
    print(f"logged info:{log_messag_stripp}")
def read_log(self):
    try:
        with open(self.log_into, 'r') as file:
            content = file. read()
        return content
    except Exception as e:
        print(f"An error occured with
reading the log file: {e}")
        return None
if __name__ == "__main":
    logger=Log("application.log")
    logger. log_into("This is an error message")
    log content = logger.read_log()
    if log_content:
        print("\nLog file content:")
        print(Log_content)
```

#28_ Create a class Config that reads configuration settings from a file and provides methods to access these settings.

```
        if params is None:
            self.cursor.execute(query)
        else:
            self.cursor.execute(query, params)
        self.connection.commit()
        return self.cursor.lastrowid
    except sqlite3.Error as e:
        print(f"An Error occured while
executing query: {e}")
def fetch_all(self, query, params):
    try:
        if params is None:
            self.cursor.execute(query)
        else:
            self.cursor.execute(query, params)
        return self.cursor.fetchall()
    except sqlite3.Error as e:
        print(f"An Error occured while
fetching data: {e}")
        return []
def close(self):
    if self.connection:
        self.connection.close()
```

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_file):
        self.db_file = db_file
        self.connection = None
        self.cursor = None
        self.connect()
    def connect(self):
        try:
            self.connection = sqlite3.connect(self.
db_file)
            self.cursor = self.connection.cursor()
            print(f"Connected to database: {self.
db_file}")
        except sqlite3.Error as e:
            print("Failed to connect to
database: {e}")
    def execute_query(self, query, params=
None):
        try:
```

```
        self.data = []
    def load_data(self):
        try:
            if not os.path.exists(self.filename):
                raise FileNotFoundError(f"The file
'{self.filename}' does not exist.")
            with open(self.filename, 'r') as file:
                self.data = [line.strip() for line
in file if line.strip()]
            print(f"Data loaded succesfuly from '{self.
filename}'")
        except FileNotFoundError as e:
            print(e)
        except IOError as e:
            print(f"An error occured while
reading the file: {e}")
    def generate_report(self):
        if not self.data:
            print("No data available to
generate a report.")
            return
        report = "===Report===\n"
        report += "\n".join(self.data)
```

```
        print(f"Connection to database
{self.db_file} closed.")
if __name__ == "__main__":
    db = Database('example.db')
    db.execute_query('CREAT TABLE IF
NOT EXISTS USERS (id INTEGER PRIMARY KEY,
name TEXT)')
    user_id = db.execute_query('INSERT
INTO USERS (name) VALUES (?)', ('Alice',))
    print(f"Inserted user with ID: {user_id}")
    users = db.fetch_all('SELECT * FROM
USERS')
    print("Users in the database:", users)
    db.close()
```

#30. 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.

```
import os
class Report:
    def __init__(self, filename):
        self.filename = filename
```

buttons.

```
import tkinter as tk
class CounterApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Counter App")
        self.counter = 0
        self.label = tk.Label(root, text=
str(self.counter))
        self.label.pack()
        self.increment_button = tk.Button(
root, text="Increment", command=self.increment)
        self.increment.pack(side=tk.LEFT, padx=20)
        self.decrement_button = tk.Button(root,
text="Decrement", command=self.decrement)
        self.decrement_button.pack(side=
tk.Right, padx=20)
    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))
```

```
        report += "\n" + "-" * 40
        return report
    def save_report(self, report, output_filename):
        try:
            with open(output_filename, 'w') as file:
                file.write(report)
            print(f"Report saved successfully to
{output_filename}.")
        except IOError as e:
            print(f"An error occured while
saving the report: {e}")
if __name__ == "__main__":
    report = Report('data.txt')
    report.load_data()
    generate_report = report.generate_report()
    if generate_report:
        print(generate_report)
        report.save_report(generate_
report, 'report.txt')
```

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

RIGHT, fill=tk.Both)
self.task_entry=tk.Entry(self.root, width=30)
self.task_entry.pack()
task_button=tk.Button(self.root, text="Add task.", command=self.remove_task()
def add_task(self):
task=self.task_entry.get()
if task!="":
self.listbox.insert(tk.End, task)
self.task_entry.delete(0, tk.END)
else:
messagebox.showwarning("Warning", "You must enter a task")
def remove_task(self):
try:
selected_task_index=self.listbox.curselection()[0]
self.listbox.delete(selected_task_index)
except:
messagebox.showwarning("Warning",

if __name__=="__main__":
root=tk.Tk()
app=CounterApp(root)
root.mainloop()
#37. Create a class ToDoApp that uses tkinter to create a todo list GUI where users can add and remove tasks
import tkinter as tk
from tkinter import messagebox
class ToDoApp:
def __init__(self, root):
self.root=root
self.root.title("To do List App")
self.root.geometry("300x400")
frame=tk.Frame(self.root)
frame.pack(pady=10)
self.listbox=tk.Listbox(frame, width="30", height="15", selectmode=tk.SINGLE)
self.listbox.pack(side=tk.pack, fill=tk.Both)
~~self.tis~~
scrollbar=tk.scrollbar(frame)
scrollbar.pack(side=tkinter.

```
        row = 1
        col = 0
        for button in buttons:
            action = lambda x=button: self.
on_button_click(x)
            tk.Button(master, text=button, padx=20,
pady=20, command=action).grid(row=row,
column=col)
            col += 1
            if col > 3:
                col = 0
                row += 1
    def on_button_click(self, char):
        if char == '=':
            try:
                result = str(eval(self.result.get()))
                self.result.set(result)
            except Exception:
                self.result.set("Error")
        elif char == 'C':
            self.result.set("")
        else:
            current = self.result.get()
```



```
"You must enter a task to remove.")
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, master):
        self.master = master
        master.title("Simple Calculator")
        self.result = tk.StringVar()
        self.entry = tk.Entry(master, text
variable=self.result, width=16, font=('Arial',
20), bd=10, insertwidth=2)
        self.entry.grid(row=0, column=0,
columnspan=4)
        buttons = ['7', '8', '9', '/',
                   '4', '5', '6', '*',
                   '1', '2', '3', '-',
                   'C', '0', '=', '+']
```

if username == "admin" and password == "password":
messagebox.showinfo("Login", "Login Successful")
else:
messagebox.showerror("Login", "Invalid Credentials")
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
from tkinter import messagebox
class WeatherApp:
def __init__(self, root):
self.root = root
self.root.title("Weather App")
self.title_label = tk.Label(root, text="Weather Information", font=('Arial', 20))

self.result.set(current + char)
if __name__ == "__main__":
root = tk.Tk()
app = CalculatorApp(root)
root.mainloop()

# 39. Creat 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
root.title("Login Form")
self.username_label = tk.Label(root, text="Username:")
self.username_label.pack(pady=5)
self.username_entry = tk.Entry(root)
self.username_entry.pack(pady=20)
def login(self):
username = self.username_entry.get()
password = self.password_entry.get()

Second Homework (built-in function)

```
        self.weather_label = tk.Label(root, text="Weather: Unknown", font=('Arial', 14))
        self.weather_label.pack(pady=20)
        self.fetch_button = tk.Button(root, text="Fetch Weather", command=self.fetch_weather)
        self.fetch_button.pack(pady=10)
    def fetch_weather(self):
        simulated_weather = "Sunny, 35°C"
        self.weather_label.config(text=f"Weather: {simulated_weather}")

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

WORLD
print(text.strip()) # Hello world
print(text.startswith("Hellow")) #output
false
print(text.endswith("!")) # output:
True
Fruit = ["apple", "banana", "cherry"]
fruit.append("orange")
print(fruit) # output: ['apple', 'banana', 'chery',
'orange']
print(frut.remove('banana')) # output =
['apple', 'chery', 'orange']
fruit.sort()
print(fruit.sort()) # output: ['apple', 'cherry',
'apple']
fruit.reverse()
print(fruit.count("cherry")) # output:
1
fruit.reverse()
print(fruit.reverse()) # output: ['orang',
'cherry', 'apple']
my_list = [1, 2, 3]
my_list.extend[4, 5], print(my_list)
#output: [1, 2, 3, 4, 5]

1- abs →>> print(abs(-10)) # output = 10
2- pow →>> print(pow(2,3)) # output: 8
3- round →>> print(round(3,14159, 2)) #output: 3.14
4- max →>> print(max(5, 58, 1)) # output: 8
5- min →>> print(min(2,6,8,11) #output: 1
6- divmod →>> print(divmod(10,3)) # output: (3,1)
7- list=[1,2,3,4,5]
print(sum(list) #output: 4
8- print(round(3,7)) # output: 4
9- import math
print(math.floor(3.9)) #output: 3
print(math.ceil(3.1)) # output: 4
text="hello world"
print(len(text)) #output: 11
print(text(capitalize)) #output: Hello World
print(replace("world", "python") # output: hello
python
print(text.split(" ")) #output: ['hellow', 'world']
words=["hellow", "world"]
print(" ".join(word)") #output: hellow world
text "Hellow, World"
print(text.lower()): #output: hellow world
print(text.upper() #output: HELLOW