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

person1 = person("Ali", 20)
print(person1.name, person1.age)
```

Ali 20

Name: Ezatullah
F-name: khadimhossain
depatment = Software Engineering

(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("How are you", person1.name)

person1 = person("Ali", 20)
person1.greet()
```

How are you Ali

(4)

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

```
Class Circle:
    def calculate (self, radius):
    return 2 * 3.14 * radius * * 2

Circle 1 = Circle()

print (Circle . calculate (3))
```

56.52

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

```
class student:
    def __init__(self, name="", grade=0, age=0):
        self.__name = name
        self.__grade = grade
        self.__age = age

    def set_name(self, name)
        self.__name = name
    def set_grad(self_grade)
        self.__grade = grade
    def set_age(self, age)
        self.__age = age

    def display_details(self):
        return self.__name, self.__grade, self.__age
```

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, amount):
        if 0 < amount < 100:
            (amount/100) * self.price
        else:
            print("the amount should be between 0 & 100")

    def display_details(self):
        return self.__brand, self.__model, self.__price
```

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 car_details(self)
        print(C1.make, C1.model, C1.year)

C1 = Car("sport", "Toyota", 2001)
C1.car_details()
```

SportToyota 2001

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

    def Manager

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

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_num, balance=0):
        self.__account_num = account_num
        self.__balance = balance

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

    def withdraw(self, amount):
        if amount < self.__balance:
            self.__balance -= amount
        else:
            print("Not enough money")

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

6. Create a bass 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 woof"

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

17

8: Create a class Account with private attribute balance. provide public methods to deposit and withdraw money.

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

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
        else:
            print("Not enough money!")

    def withdraw(self, amount):
        if amount > 0 and amount < self.__balance:
            self.__balance -= amount
        else:
            print("Something bad happened")

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

19

9:

(5) create a class Rectangle with methodes to compute the area and perimeter. Initialize the class with the length and width

```
class Rectangle:
    def area_calculate(self, length, width):
    self.length = length
    self.width = width
    return "The area of rectangle is: " self.length * self.width

    def perimeter_calculate(self, length, width)
        self.length = length
        self.width = width
        return "the perimeter of rectangle is:" 2* self.length + 2* self.width

rec1 = Rectangle()
print(rec1.area_calculate(2,5))
print(rec1.perimeter_calculate(2,5)
```

The area of rectangle is: 10
the perimeter of rectangle is: 14

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

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

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

class Penguin(Bird):
    def fly(self):
        return "Penguin can not fly"
```

12

9: Create a class Book with privat attributes ~~brand~~ title, author, and pages. provide public methodes to get and set these attributes.

```
class Book:
    def __init__(self, title="", author="", pages=0):
        self.__title = title
        self.__author = author
        self.__pages = pages

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

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

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

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

7:

Create a base class Shape with a method area. Create derived classes Square a 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 ** 2

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

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

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 can not fly.

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

class Bike(vehicle):
    def drive(self):
        return "riding the bike)

class Truck(vehicle):
    def drive(self):
        return "driving the truck)
```

20:

Create a class Zoo with attribute 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 = []

    def add_animal(self, animal):
        self.animals.append(animal)

    def remove_animal(self, animal):
        if animal in self.animals:
            self.animals.remove(animal)
        else:
            print("it doesn't exist")

    def animal_list(self):
        return self.animals
```

17: Create a class School with attribute 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 = []

    def add_Student(self, student):
        self.students.append(student)

    def remove_Student(self, student):
        if student in self.students:
            self.students.remove(student)
        else:
            print("it hasn't exist")

    def student_list(self):
        return self.students
```

16:

Create a class Library with attributes name and books (a list of book object). provide methodes to add and remove books.

```
class Library:
    def __init__(self, name, books):
        self.name = name
        self.books = []
    def add_book(self, book):
        Self.books.appened(book)

    def remove_book(Self, book):
        if book in self.books:
            self.book.remove(book)
        else:
            print("it has not exist")

    def book_list(self):
        return self.books
```

17:

27: Create a class ShoppingCart with methods to add and 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.item=[]

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

    def remove_item(self, item):
        self.item
        if item in self.item:
            Self.item.remove(item)
        else:
            print("does not exist")

    def display_item(self):
        return self.item
```

19:

Create a class Company with attributes name and employees (a list of Employe objects). provide methodes to add and remove employees.

```
class Company:
    def __init__(self, name, employees):
        self.name = name
        self.employees = []

    def add_employee(self, employee):
        self.employees.append(employee)

    def remove_employee(self, employee):
        if employee in self.employee:
            self.employee.remove(employee)
        else:
            print("it has not exist")

    def employee_list(self):
        return(self.employees)
```

18:

Create a class Team with attributes name and members (a list of person objects). provide methodes to add and remove members.

```
class Team:
    def __init__(self, name, members):
        self.name = name
        self.members = []

    def add_member(self, member):
        self.members.append(member)

    def remove_member(self, member):
        if member in self.member:
            self.member.remove(member)
        else:
            print("it has not exist")

    def member_list(self):
        return self.members
```

19:

87

Create a class TodoApp that uses tkinter to create a to-do list GUI where 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")

        self.task_entry = tk.Entry(root, width=40)
        self.task_entry.pack(pady=10)

        self.add_button = tk.Button(root, text="add tasks",
                                    command = self.add_task)
        self.add_button.pack(pady=5)

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

28/ Create a class Restaurant with attributes name and menu(a list of item objects). provide methodes to add and remove items from the menu.

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

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):
        if item in self.menu:
            self.menu.remove(item)
        else:
            print("item doesn't exist")
```

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

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

    def display_details(self):
        print("The movie name is:" self.name)
        print("The seat number is:" self.seat_num)
        print("The movie's price is: self.price)

    def apply_discount(self, discount)
            self.price -= self.price * (discount / 100)
            print(f"The price after discount is: {self.price}")
```

88.

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("Simple calculator")
        self.display = tk.Entry(root, width=35,
                borderwidth=5, font=("Arial",18),
                        justify='right')
        self.display.grid(row=0, columnspan=4,
                        padx=10, pady=10)

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

```
def calculate_result(self):
    try:
        expression = self.display.get()
        result = eval(expression)
        self.clear_display()
        self.display.insert(tk.END, str(result))
    except Exception as e:
        self.clear_display()
        self.display.insert(tk.END, "error")

def clear_display(self):
    self.display.delete(0, tk.END)

if __name__ == '__name__':
    root = tk.Tk()
    app = calculatorApp(root)
    root.mainloop()
```

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.count = 0
        self.root = root
        self.root.title("Counter app")
        self.label = tk.Label(root, text=str(self.count),
                              font=("Arial", 48))
        self.label.pack(pady=20)
        self.increment_button = tk.Button(root,
                              text="increase", command=self.increment)
        self.increment.pack(side="left", padx=20)
        self.decrement_button = tk.Button(root,
            text="decrease", command=self.decrement)
        self.decrement_button.pack(side="right", padx=
```

Q) Create a class Hotel with attributes name and room (a list of Room object). Each room should have attributes room_number and is_occupied. provide methods to book and check_out room.

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

    def book_room(self, room_number):
    def add_room(self, room):
        se
```

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

    def book_room(self, room_num
    def add_room(self, room)
        self.rooms.append(room)
```

```
self.remove_button = tk.Button(root, text="remove task",
                               command=self.remove_task)
self.remove_button.pack(pady=5)

def add_task(self):
    task = self.task_entry.get()
    if task:
        self.task_listbox.insert(tk.END, task)
        self.task_entry.delete(0, tk.END)
    else:
        messagebox.showwarning("error", "please enter a task")

def remove_task(self):
    selected_task_index = self.task_listbox.curselection()
    if selected_task_index:
        self.task_listbox.delete(selected_task_index)
    else:
        messagebox.showwarning("error", "please chose a task")

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

```
def increment(self):
    self.count += 1
    self.update_label()

def decrement(self):
    self.count -= 1
    self.update_label()

def update_label(self)
    self.label.config(text=str(self.count)

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

```
def book_room(self, room_number):
    for room in self.rooms:
        if room.room_number == room_number
                and not room.is_occupied:
            room.is_occupied = True
            print("The room was booked")


def cheak_out_room(self, room_number):
    for room in self.rooms:
        if room.room_number = room_number
            and room.is_occupied:
            room.is_occupied = False
            print("the room cheak outed")


def display_rooms(self):
    return self.rooms
```

29/

Create a class Flight with attributes flight-number, destinition and passengers (a list of person object). provid methods to add & remove passengers.

```
class Person:(self, name, passport-number):
    def __init__(self, name, passport-number):
        self.name = name
        self.passport-number = passport_number

class Flight:
    def __init__(self, flight-number, destinition):
        self.high-number= flight-number
        self.destinition = destinition
        self.passengers =[]

    def add-passenger(self, person):
        self.passengers.append(person)

    def remove-passengers(self, person):
        if person in self.passenger:
            self.passengers.remove(person)
        else:
            print("person does not exist")
```

```
buttons = []
for i, text in enumerate(button_text):
    button = tk.Button(root, text=text, padx=20, pady=20,
                       font=("Arial", 18),
                       command=lambda t=text: self.on_button_click(t))
    buttons.append(button)
    row = (i // 4) + 1
    column = i % 4
    button.grid(row=row, column=column)

clear_button = tk.Button(root, text='C', padx=20, pady=20,
                         font=("Arial", 18), command=self.clear_display)
clear_button.grid(row=5, column=0, columnspan=4)

def on_button_click(self, text):
    if text == "=":
        self.calculate_result()
    else:
        self.display.insert(tk.END, text)
```