تاريخ: / / موضوع:

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", 30)
print(Ahmad.name, Ahmad.age)
```

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

```
class Person():
    def __init__(Self, name, age):
        Self.name = name
        Self.age = age
    def greet(Self, greet):
        return f"{greet} {self.name}"
Ahmad = Person("Ahmad", 30)
Print(Ahmad.greet("Hi"))
r1 = Rectangle(15, 7)
r1.area()
```

CamScanner

موضوع: _____ تاریخ: / /

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

```
class car():
    def __init__(self, make, model, year):
        Self.make = make
        Self.model = model
        Self.year = year
    def detail(self):
        print(f"{self.make} {self.model} {self.year}")
Pride = Car("Iranian", "new", 2020)
Pride.detail()
```

④ Create a class Circle with a methode to compute the area. Initilize the class with the radies.

```
class Circle():
    radius = 16
    def compute(self):
        print("the area is:", Circle.radius * 0.5)
circle_area = Circle()
Circle_area.compute()
```

CamScanner

تاريخ: / / موضوع:

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

```
class Rectangle():
    def __init__(self, width, length):
        self.with = width
        self.length = length
    def area(self):
        print(self.width * self.length)
    def perimeter(self):
        print(2(self.width + self.length))
r1 = Rectangle(15,7)
r1.area()   r1.perimeter()
```

(6) create a base class Animal with a mithod speak. Create two derived classes dog and Cat that overide the speak method.

```
class Animal():
    def __init__(self, an_type)
        self.an_type = an_type
    def speak(self):
        print(f'the {self.an_type} is speaking!')
```

CamScanner

```
class Dog(Animal):
    pass
class Cat(Animal):
    pass
dog = Dog("dog")
dog.speak()
```

⇒

```
cat = Cat("cat")
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 __init__(self)
        pass
    def area(self):
        print("calculating the area")
class Square(Shape):
    pass
class Triangle(Shape):
    pass
square = square(area)
square = square(), trian = Triangle()
square.area(), trian.area()
```

CamScanner

⑧ create a class Employee with attributes name and salary. create a derived class Manager with an aditional 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
mgr1 = Manager("Mansor", 8000, 'Computer')
```

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):
        Print("driving!")
class Bike(vehicle):
    pass
class Truck(vehicle):
    pass
```

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

CamScanner

(10) Create a base class Bird with a method fly. Create derived classes Eagle and penguin. override the fly method in penguin to indicate penguin can't fly.

```
class Bird():
    def fly(self):
        print("flying!")
class Eagle(Bird):
    pass
class penguin(Bird):
    def fly(self):
        print("penguin can't fly").
peng = Penguin() ——> Peng.fly()
```

(IV) — Encapsulation —

(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):
        return self.__balance + amount
```

CamScanner

موضوع: _____ تاريخ: / /

```
    def withdraw(self, amout):
        return self.__balance - amount
z=Account(80)
Print(z.deposit(9))
Print(z.withdraw(9))
```

(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 setter(self, title, author, pages):
        self.__title = title
        self.__author = author
        self.__pages = pages
    def getter(self):
        print(f'{self.__title}, {self.__author} {self.__pages}')
book= Book('novel', 'habib', 809)
book.setter("life", "ali", 30)
book.getter()
```

CamScanner

موضوع: _____ تاريخ / /

(13) Create a class Laptop with private attributes brand, model and price. Provide public method to display 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):
        if( discount == "9851"):
            v = self.__price * 0.95
            return f'{v} this is after applying discount'
    def detail(self):
        return f'{self.__brand}{self.__model}{self.__pr
dell = Laptop("Dell", 6490, 899)
print(dell.discount("9851")
Print(dell.detail())
```

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

موضوع: _____ تاريخ: / /

```
class BankAccount():
    def __init__(self, account_number, balance):
        self.__account_number = account_number
        self.b_balance = balance
    def deposit(self, deposit_amount):
        V = self.__balance + deposit_amount
        print(V)
    def withdraw(self, withdraw_amount):
        V = self.__balance - withdraw_amount
        print(V)
    def checkBalance(self):
        print(f'{account_number} {balance}')
Person1 = BankAccount(200, 900)
Person1.deposit(100)
person1.withdraw(100)
Person1.checkBalance()
```

(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 detail.

```
class Student():
    def __init__(self, name, grade, age):
        self.__name = name
        self.__grade = grade
        self.__age = age
    def setter(self, name, grade, age):
        self.__name = name
        self.__grade = grade
        self.__age = age
    def getter(self):
        print(f'{self.__name} {self.__grade} {self.__age}')
    def display(self):
        print(f'Name: {self.__name}, grade: {self.__grade}')
Ali = Student("Ali", "Ten", 20)
Ali.setter("Ali", "Ten", 20)
Ali.getter()
Ali.display()
```

— Advance Concept of OOP —

(16) create a class Library with attributes name and books [a list of Book objects]. provide methods to add and remove books.

موضوع: _____ تاريخ: / /

```
class Library():
    def __init__(self, name):
        self.name = name
        self.books = []
    def adder(self, book):
        if book not in self.books:
            self.books.append(book)
    def remover(self, book):
        if book in self.books:
            self.books.remove(book)
eqra = Library("math")
eqra.adder("math")
eqra.remover("math")
```

(17) Create a class School with attributes name and students (a list of students). provide methodes to add and remove students.

```
class School():
    def __init__(self, name):
        self.name = name
        self.students = []
```

موضوع: _____ تاریخ: / /

```
def add_st(self, name):
    if name not in self.students:
        self.students.append(name)
def rem_st(self, name):
    if name in self.students:
        self.students.remove(name)
Homayon = School("Hoya")
Homayon.add_st("Ali") / Homayon.rem_st("Ali")
```

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

```
class Team():
    def __init__(self, name):
        self.name = name
        self.members = []
    def add_st(self, member):
        if (member not in self.members):
            self.members.append(member)
    def rem_st(self, member):
        if (member in self.members):
            self.members.remove(member)
```

تاريخ: / / موضوع:

reate a class Company with attributes name and
ployees (a list of Employees). provide methode
add and remove employees.

```
ss company():
    def __init__(self, name):
        self.name = name
        self.emps = []
    def adder(self, emp):
        self.emps.append(emp)
    def remover(self, emp):
        self.emps.remove(emp)
```

20) create a class Zoo with attributes name and animals(
list of animal). provide methods to add and remove.

```
class Zoo():
    def __init__(self, name):
        self.name = name
        self.animals = []
    def adder(self, animal):
        if (animal not in animals)
            self.animals.append(animal)
    def remove(self, animal):
        if (animal in self.animals)
            self.animals.remove(animal)
```

تاريخ: / / موضوع: _____

(21) create a class FileManger with methods to read from and write to a file.

```
class FileManager():
    def reading(self, File):
        with open(f'{File}', 'r') as f:
            print(f.read())
    def writing(self, file):
        with open(f'{file}', 'w') as f:
            print(f.write("This is one"))
```

(22) Create a class log with methods to write error message to a log file.

```
class Log():
    ~~def __init__(self, name)~~:
    def error_massages(self):
        with open("log_Eror.txt", Y)
            print("your passward is wrong!")
Ali = log()
Ali.error_massages()
```

تاريخ: / / موضوع: _____

(23) create a class flight that reads configuration or setting at setting from a file and provides methods to access those setting:

```
class Read(self):
    with open("setting_file", "r"):
def access(self):
    with open("setting_file", "a") as f
        print(f.__dir__())
```

(24) create a class Database that connects to a database and provides methods to execute queries. Handle exception if the connection fails.

```
import sqlite3
class Database():
    def connection(self):
        try:
            conn = sqlite3.connect("mydata.db")
            c = conn.cursor()
            c.execute("""CREATE TABLE students(
                name text, last text, age integer)""")
```

```
    c.execute("INSERT INTO students VALUES('ali',
            'abbasi', 20)")
    c.execute("SELECT * FROM students")
    print(c.fetchone())
    conn.commit()
    conn.close()
except:
    print("The connection was failed!")
```

(25) Create a class Report that generates a report from data in a file.

```
class Report:
    def generater_report("data_file", "r") as f:
        with open(f.readlines()):
    def read(self):
        try:
            with open("data_file", "r") as f:
                print(f.readlines())
        except:
            print("Not exists")
A=Report()
print(A.read())
print(A.generater_report)
```

(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 discount.

```
class Ticket():
    def __init__(self, moviename, seatnumber, price):
        self.movie_number = movie_number
        self.seat_number = Seat_number
        self.price = price
    def discount(self, code):
        if code == "9812":
            print(self.price * 0.95, "discount amount")
    def detail(self):
        print(f'{self.movie_numb} {self.seat_number} {price}')
```

(27) creat a class shopping cart with methods to add and removes items.

```
class ShopCard():
    def __init__(self, items):
        self.items = items
    def adder(self, item):
        self.items.append(item)
    def remove(self, item):
        self.remove(item)
```

موضوع: _____ تاريخ / /

(28) Creat a class Restaurant with attributes name and menu (a list of item objects). provide methods to add and remove items from menu.

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

(29) create a class Flight with attributes fli_num, destinatio and passengers (a list of people). provide methods to add and remove passenger.

```
class Flight():
    def __init__(self, flight_number, destination):
        self.flight_number = flight_num
        self.destination = self.destination
        self.passengers = []
    def adder(self, passen):
        self.passengers.append(passen)
    def remove(self, passen):
        self.passenger.remove(passen)
```

(30) create a class Hotel with attributes name and rooms (a list of rooms). Each rome should have attributes room_number and is_occupied. provide methods to book and check rooms.

```
class Room():
    def __init__(self, room_number: int, is_occupied: bool):
        self.room_number = room_number
        self.is_occupied = is_occupied
room1 = Rom(1, True)
room2 = Room(2, False)
class Hotel():
    def __init__(self, name):
        self.name = name
        self.room = [room1, room2]
    def check(self, room_num: int):
        if self.room[room_num -1].is_occupied == True:
            print('Room is occupied')
        else: print('Room is free')
    def book(self, room_num: int):
        if self.room[room_num -1].is_occupied == True :
            Print('It is already booked')
        else:
            Print('You are welcome')
```

(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_value = 0
        self.label = tk.Label(root, text=f"Counter: {self.counter_value}
        Font = ("Helvetica", 18))
        self.label.pack(pady=20)
        self.increment_button = tk.Button(root, text="increment",
        command = self.increment_counter)
        self.increment_button.pack()
    def increment_counter(self):
        self.counter_value += 1
        self.label.config(text=f"counter: {self.counter_value}")
    def decrement_counter(self):
        self.counter_value -= 1
        self.label.config(text=f"Counter: {self.counter_value}")
root = tk.Tk()
root.title("CounterApp")
~~root.geography~~ root.geometry('600x400')
app = CounterApp(root)
root.mainloop()
```

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

```
import tkinter as tk
from tkinter import messagebox
class ToDoApp:
    def __init__(self, root):
        self.root = root    self.title ("TODOApp")
        self.tasks = []
        self.task_entry = tk.Entry(root, width=40)
        self.task_entry.pack(pady=10)
        add_button = tk.Button(root, text="Add", command=self.add_task)
        add_button.pack
        self.task_listbox = tk.Listbox(root, height=15, width=50)
        self.task_listbox.pack(pady=10)
        remove_button = tk.Button(root, text="Remove", command=
        remove_button.pack()                                  self.remove_task)
    def add_task(self):
        task = self.task_entry.get().strip()
        if task:
            self.tasks.append(task)
            self.task_listbox.insert(tk.END, task)
            Self.task_entry.delete(0, tk.END)
        else:
            messagebox.showwarning("please enter a task")
```

(38) creating calculator

```
import tkinter as tk
class CalculatorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("calculator")
        self.entry = tk.Entry(root, width=20, borderwidth=5))
        self.entry.grid(row=0, column=0, columspan=4, pady=10)
        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) ]
For (text, row, col) in buttons:
    button = tk.Button(root, text=text, pady=20 )command=
            lambda t=text: self.handle_click(t))
    button.grid(row=row, column=col, padx=5, pady=5)
def handle_click(self, text):
    current = self.entry.get()
    if
root = tk.Tk()
app = CalculatorApp(root)
root.mainloop()
```

(39) creating a login page with error message

```
import tkinter as tk  from tkinter import messagebox
class LoginApp:
    def __init__(self, root):
        self.root=root
        self.root.title("Login")
        self.username_label =tk.Label(root, text="Username")
        self.username_label.grid(row=0, colum=0, padx=10,pady=10)
        self.password_label =tk.Label(root, text="password")
        self.password_label.grid(row=1, column=0, padx=0)
        self.password_entry = tk.Entry(root, show='*')
        self.login_button = tk.Button(root, text="login", command=
                                                    self.check_credi
        self.login_button.grid(row=2.column=0, padx=10)
def check_credentials(self):
        username = self.username_entry.get()
        password = self.password_entry.get()
        if username == "admin" and password = "password":
            messagebox.showinfo("login succeed")
        else: messagebox.showerror("failed")
root=tk.Tk()
app = LoginApp(root)
root.mainloop()
```

(40) creating weatherApp by tkinter

```
import tkinter as tk
from tkinter import messagebox
class WeatherApp():
    def __init__(self,root):
        self.root = root
        self.root.title("weather App")
        self.city_entry = tk.Entry(root, width = 30)
        self.city_entry.pack(pady=10)
        self.get_weather_button = tk.Button(root, text= "weat",
                                            command= self.get_weather)
        self.get_weather_button.pack()
        self.weather_label = tk.Label(root, text="")
        self.weather_label.pack(pady=20)
def get_weather(self):
    city = self.city_entry.get()
    if not city:
        messagebox("warning")
    try:
        response = requests.get(url)
    Excep: print("Not found")
root = tk.Tk()
app = WeatherApp(root)
root.mainloop()
```