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

person1 = Person1("Alijan", 21)
print(f"Name: {person1.name}, Age: {person1.age}")
```

2 greeting

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

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

person1 = Person("Rohullah", 20)

person1.greet()
print(person1.greet())
```

3

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

    def car_details(self):
        print(f"Car Details: {self.year} {self.make} {self.model}")

car1 = Car("Lamborgini", "Toyota", 2022)

car1.car_details()
```

4

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

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

circle1 = Circle(5)

print(f"Area of the circle: {circle1.area():.2f}")
print("Area of the circle: {circle1.area():.2f}".__Format__)
```

5

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

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

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

rect1 = Rectangle(6, 7)
print(f"Area of the rectangle: {rect1.area()}")
print(f"Perimeter of the rectangle: {rectangle1.perimeter()}")
```

result 42

result 26

5 Create a class Employee with attribute name and salary. Create a derived class Manager with an additional attribute department

6 Create a base class Animal with a method speak. Create two drived classes Dog and cat that override the speak method.

```
class Animal:
    def speak(self):
        return f"{self} Animals can speak like human!"

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

~~Haiwan = Animal()~~
~~sag = d~~
class Dog(Animal):
    def speak(self):
        return "Wag Wag"

Haiwan = Animal()
Sag = Dog()
Peshag = Cat()
print(Haiwan.speak())
print(~~Haiwan~~ Sag.speak())
print(peshag.speak())
```

7 create a base class shape with a method area. Create derived classes square an 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

if __name__ = "__main__":
    square = Square(4)
    triangle = Triangle(3,5)
    print(f"Area of the Square {square.area()}")
    print(f"Area of the tringle: {triangle.area()}")
```

8 Create a class Employee with attribute name and sallary. Create a derived class Manager with an additional attribute department.

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

class Minager(Karmand):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

emp = Employee("Ali", 30 as)
emp1 = Manager("Ahmad", 2401, "ISE")
print(emp.name)
print(emp.salary)
print(Ahmad.deparments)
```

9 Create a base class vehicle with a method drive. create classes Bike and Truck that override the drive method

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

class Bike(Vehicle):
    def drive(self):
        return "Bike is being driven by me."

class Truck(Vehicle):
    def drive(self):
        return "Truck isn't being drived."

mysical = Bike()
comuon = Truck()
print(mysical.drive())
print(comuon.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 high."

class penguin(Bird):
    def fly(self):
        return "penguin cannot fly."

Par = Bird()
Oqab = Eagle()
Pan = penguin()
print(par.fly()) → Noun
```

11. Create a class Account with private attribute balance. Provide public method to
deposit and with draw money

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

    def deposit(self, amounth):
        self.balance += amounth
        print("Deposit is successful. your new blance is", self.balance)

    def withdraw(self, amounth):
        if amounth > self.balance:
            print("insufficient balance")

        else:
            self.balance -= amounth
            print("withdrawl is successful new balance is", self.balance)

    def __show_balance(self):
        self.__balance(self)
        print("your current balance is", self.__balance)

account = Account(5000)
account.deposit(22500)
account.withdraw(10000)
```

12: Create a class Book with private attributes title, author, and page. Provide
public method 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):
        print("title is", self.__title)

    def get_author(self):
        print("author name is", self.__author)

    def get_pages(self):
        print("pages of book is", 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("thinker", "Qorban", 530)
book.get_title()
book.get_author()
book.get_pages()
book.set_title("new title thinker")
book.set_author("new author is Ali")
book.set_pages("new pages 630")
print(book.get_title())
print(book.set_author())
print(book.set_pages())
```

13 create a class student with private attributes name, grade, and age, provide

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_percentage):
        if 0 < discount_percentage <= 100:
            discount_amount = (discount_percentage / 100) * self.price
            self.price -= discount_amount
            print(f"Discoun applied, {discount_percentage}%. New price is
                  ${self.price:}")
        else:
            print("Discount percentage must be between 0 and 100.")

    def display_details(self):
        """Display the details of the laptop."""
        print(f"Brand, {self.brand}")
        print(f"Model: {self.model}")
        print(f"price: $ {self.price:}")
laptop = Laptop("mac", "XPS 13", 400, "$")
laptop.display_details()
laptop.apply_discount(10)
laptop.display_details()
```

14 create a class BankAccount with private attributes accoun_number and balance. provide method to deposit, withdraw, and check the balance.

```
class Bank Account:
    def __init__(self, account_number, initial_balance):
        self.__account_number = account_number
        self.__initial_balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            amount += self._initial_balance
            print(f"Deposite amount is {amount}, New, balance {self._initial_balance}
                                                                                    }

        else:
            print("amount must be positive")

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.__initial_balance:
                self.__initial_balance -= amount
                print(f"withdraw seccesful {amount}, your current {self_initial_balance}
                                                                                    ")

        else:
            print("your don't have insufficient, plase deposit your Account then try again")

    def check balance(self):
        print(f"your current balance is {self.initial_balance}")

my account = Account
             Bank Account(1213, 3500)
my account.checkbalance()
my account.deposit(500)
my account.withdraw(1000)
my accounts.withdrow(10000)
```

```
# 15 Create a class student with private attributes name, grade, and age, provide
# method 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):
        if age > 0:
            self.__age = age
        else:
            print("Age must be positive.")

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

student = Student("ali", "10", 15)
student.display_details()

student.set_name("Rohullah")
student.set_grade("9")
student.set_age(16)
student.display_details()
# Display updated student details
student.display_details()
```

16 create a class library with attributes name and book (a list of Book objects)
provide method to add and remove books.

```
class Book:
    def __init__(self, book_name):
        self.book_name = book_name

    def __str__(self):
        return F"Book- name is: {self.book-name}"


class library:
    def __init__(self, name):
        self.name = name
        self.books = []

    def add_book(self, book):
        """Add a book to the library."""
        if isinstance(book, Book):
            self.books.append(book)
            Print(F" Added book: {book}")

        else:
            print(f"only objects of type Book can be added.")

    def remove_book(self, book_name):
        """Remove a book from the library by title."""
        for book in self.books:
            if book.book_name == book_name:
                self.book.remove(book)
                print(f" Removed book: {book}")
                return
            Print(f"no book found with title '{title}'.")

    def display_books(self):
        """Display all book in the library."""
        if self.books:
            print(f" Books in {self.name}:")
            for book in self.books:
                print(book)
library = Library("Kateb")
book1 = Book(" C Sharp programing")
book2 = Book("Jave")
library.add_book(book1)
library.add_book(book2)
library.display_books()
library.remove_book(" C sharp programing")
```

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):
        self.name = name

    def __str__(self):
        return f"Name: {self.name}"

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

    def add_student(self, student):
        if isinstance(student, Student):
            print(f"Added student: {student}")

    def remove_student(self, name):
        for student in self.students:
            if student.name == name:
                self.students.remove(student)
                print(f"Removed student: {student}")

    def display_students:
        if self.students:
            print(f"Stadens in {self.school_name}:")
            for student in self.students:
                print(student)

school = School("Shirday High School")
student1 = Student("Ali Reza")
student2 = Student("Ali Jan")
student3 = Student("Eid Mohammad")

school.add_student(student1)
school.add_student(student2)
school.add_student(student3)

school.display_students()
school.remove Student("Ali")
school.display_students()
```

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

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __str__(self):
        return f"Name: {self.name}, Age: {self.age}"

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

    def add_member(self, person):
        """Add a member to the team."""
        if isinstance(person, Person):
            self.members.append(person)
            print(f"Added member: {person}")
        else:
            print("Only objects of type Person can be added.")

    def remove_member(self, name):
        """Remove a member from the team by name."""
        for person in self.members:
            if person.name == name:
                self.members.remove(person)
                print(f"Removed member: {person}")
                return
        print(f"No member found with name '{name}'.")

    def display_members(self):
        """Display all members of the team."""
        if self.members:
            print(f"Members of {self.name}:")
            for person in self.members:
                print(person)
        else:
            print(f"No members available in {self.name}.")

team = Team("Developers")
person1 = Person("Alice", 30)
person2 = Person("Bob", 25)
person3 = Person("Charlie", 35)

team.add_member(person1)
team.add_member(person2)
team.add_member(person3)
team.display_members()
team.remove_members("Bob")
team.display_members()
```

19 create a class company with attributes name and employees (a list # of Employee objects). provide method to add and remove employees.

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

    def __str__(self):
        return f"Name : {self.name}, Age : {self.age}, Salary : {self.salary}"

class company:
    def __init__(self, company_name):
        self.company_name = company_name
        self.employees = []

    def add_employee(self, employee):
        if isinstance(employee, Employee):
            self.employees.append(employee)
            print(f"Added Employe is : {employee}")
        else:
            print("only object Employee can add")

    def remove_employee(self, name):
        for employee in self.employees:
            if employee.name == name:
                self.employees.remove(employee)
                print(f"Removal employee is : {employee}")

    def display_details(self):
        if self.employees:
            print(f"Employees of : {self.company_name}")
            for employe in self.employees:
                print(employee)

Mic = company("Microsoft")
employee1 = Employee("Mark", 25, 2500)
employee2 = Employee("Putin", 65, 30000)
employee3 = Employee("Obama", 60, 40000)

Mic.add_employee(employee1)
Mic.add_employee(employee2)
Mic.add_employee(employee3)
Mic.remove_employee("Mark")
Mic.display_details()
```

```
## 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, species, age):
        self.name = name
        self.species = species
        self.age = age

    def __str__(self):
        return f"Name: {self.name}, Species: {self.species}, Age: {self.age}"

class Zoo:
    def __init__(self, name):
        self.name = name
        self.animals = []

    def add_animal(self, animal):
        """Add an animal to the Zoo."""
        if isinstance(animal, Animal):
            self.animals.append(animal)
            print(f"Added animal: {animal}")
        else:
            print("Only objects of type Animal can be added.")

    def remove_animal(self, name):
        """Remove an animal from the Zoo by name."""
        for animal in self.animals:
            if animal.name == name:
                self.animals.remove(animal)
                print(f"Removed animal: {animal}")
                return
        print(f"No animal found with name '{name}'.")

    def display_animals(self):
        """Display all animals in the Zoo."""
        if self.animals:
            print(f"Animals in {self.name}:")
            for animal in self.animals:
                print(animal)
        else:
            print(f"No animals available in {self.name}.")

zoo = Zoo("Wildlife Safari")
animal1 = Animal("Lion", "Panthera Leo", 5)
animal2 = Animal("Elephant", "Loxodonta africana", 10)
animal3 = Animal("Giraffe", "Giraffa camelopardalis", 7)
zoo.add_animals(animal1)
zoo.add_animals(animal2)
zoo.add_animals(animal3)
zoo.display_animals()
zoo.remove_animal("Elephant")
zoo.display_animals()
```

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

```
class FileManager:
    def __init__(self, Manager_name):
        self.man_name = Manager_name

    def write_to_File(self, content):
        print(f" the File is writing by {self.man_name} ")

    def read_from_file(self):
        print(f" the file is reading by {self.man_name}
```

```
File_manager = FileManagger ("Manager_Abdullah")

File_manager.write_to_File('This is an example content.')

content = File_manager.read_from_File()
If content is not None:
    print("Content read from file:")
    print(content)
```

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

```
import os
From datetime import datetime

class Log:
    def __init__(self, log_file):
        self.log_file = log_file

    def _write_message(self, message):
        print(f"Error writing to log file: {message}")

    def log_error(self, message):
        formatted_message = f"ERROR: {message}"
        self._write_message(formatted_message)

    def log_warning(self, message):
        formatted_message = f"WARNING: {message}"
        self._write_message(formatted_message

log = Log('application.
log.log_error('This is an error message.')
log.log_warning('This is a warning message.')
```

```
## 23. Create a class Config that reads configuration settings from a file and
#  provids method to acces thess settings.

class Config:
    def __init__(self, config_file):
        self.config_file = config_file
        self.settings = self._load_config()

    def _load_config(self):
        print(f"configuration file not foand: {self.config_file}")

    def get(self):
        return f"in this program error {self.config_file}"

    def set(self):
        self.settings = {}
        self._save_config()

    def _save_config(self):
        print(f"Error writing to the configuration file: {self.config_file}")

config = Config('config.json')
print("Database Host:", config.get())
print(config.set())
print(config.get())
```

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

```
import sqlite3
from sqlite3 import Error

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("Connection to database established successfully.")

        except Error as e:
    def execute_query(self, query, params=()):
        try:
            self.cursor.execute(query, params)
            self.connection.commit()
            print("Query executed successfully.")

        except Error as e:
            print(f"Error executing query: {e}")

    def fetch_all(self, query, params=()):
        try:
            self.cursor.execute(query, params)
            return self.cursor.fetchall()

        except Error as e:
            print(f"Error fetching data: {e}")

    def close(self):
        if self.connection:
            self.connection.close()
            print("Database connection closed.")

db = Database('example.db')
db.execute_query('''
    CREATE TABLE IF NOT EXISTS users(
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT NOT Null,
        email TEXT NOT Null.
''');
```

```
db.execute_query('INSERT INTO users (username, email) VALUES (?, ?)', ('John_doe',
                                                    'John@example.com'))
users = db.fetch_all('SELECT * FROM users')
for user in users:
    print(user)

db.close()
```

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

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

    def _read_file(self):
        if not os.path.isfile(self.file_path):
            raise FileNotFoundError(f"File does not exist: {self.file_path}")

        try:
            with open(self.file_path, 'r') as file:
                return file.readlines()
        except IOError as e:
            raise IOError(f"Error reading file: {e}")

    def generate_report(self):
        try:
            data = self._read_file()
            report = self._process_data(data)
            return report
        except Exception as e:
            print(f"Error generating report: {e}")

    def _process_data(self, data):
        num_lines = len(data)
        report = f"Report Generated\nNumber of lines: {num_lines}\n\n"
        report += "File content:\n" + "".join(data)
        return report

report = Report('data.txt')
report_content = report.generate_report()
if report_content:
    print("Generated Report:\n")
    print(report_content)
```

26 create a class Ticket for a movie theater with attributes movie_name, seat_number # and price. provide methods to display ticket details, 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 apply_discount(self, discount_percentage):
        if 0 <= discount_percentage <= 100:
            discount_amount = (discount_percentage / 100) * self.price
            self.price -= discount_amount
            print(f"Discount applied: {discount_percentage}%")
        else:
            print("Invalid discound percentage. It must be between 0 and 100.")

    def display_details(self):
        details = (
            f"Movie Name: {self.movie_name}\n"
            f"Seat Number: {self.seat_number}\n"
            f"price: $ {self.price:.2f}"
        )
        print(details)

ticket = Ticket("The Matrix", "A12", 20, 10)
ticket.apply_discount(10)
ticket.display_details()
ticket.display_d
ticket.display_discount()
ticket.apply_discount(10)
ticket.display_details()
```

```
## 27 Create a class shopping cart with attributes name, display
# 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 __repr__(self):
        return f"{self.name} (${self.price:.2f})"

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

    def add_item(self, item):
        if isinstance(item, Item):
            self.items.append(item)
            print(f"Added {item} to the cart.")
        else:
            print("only Item objects can be added to the cart.")

    def remove_item(self, item_name):
        for item in self.items:
            if item.name == item_name:
                self.items.remove(item)
                print(f"Removed {item} from the cart.")
                return item_nam
        print(f"Item '{item_name}' not found in the cart.")

    def display_items(self):
        if not self.items:
            print("The cart is empty.")
        else:
            print("Shopping cart Items:")
            for item in self.items:
                print(item)

item1 = Item("Apple", 0.99)
item2 = Item("Bread", 2.49)
item3 = Item("Milk", 1.89)
cart = ShoppingCart()
cart.add_item(item1)
cart.add_item(item2)
cart.display_items()
cart.remove_item("Apple")
```

28 create a class Restaurant with attributes name and menu

```
class Restaurant:
    def __init__(self, name):
        self.name = name
        self.costomer = []
    def add_student(self, costomer_name):
        self.costomer.append(costomer_name)
        print(f"Book {costomer_name} added")
    def remove_student(self, student_name):
        if student_name in self.costomers:
            self.costomers.remove(student_name)
            print(f"student {student_name} removed")
        else:
            print("No student in school")
    def school_details(self):
        if self.costomers:
            print("student in school,")
            for stu in self.costomers:
                print(f"- {stu}")
        else:
            prins("No student in school")

school = Restaurant("Namona")
school.add_student("Ahmad")
school.add_student("Javad")
school.remove_student("Ali")
school.remove_student("Javad")
school.school_details()
```

29 create a class filight with attributes flight_number, destination, and passengers (a list of person objects).

```
class filight:
    def __init__(self, flight_name, destination):
        self.fl_name = flight_name
        self.destination = destination
        self.pasenger []
    def add_passerger(self, pasenger_name):
        self.pasenger.append(pesenger_name)
        print(f"Book {passenger_name} added")
    def remove_pasenger(self, student_name):
        if student_name in self.pesenger:
            self.pasenger.remove(student_name)
            print(f"pasenger {student_name} removed")
        else:
            print("No pasenger in fright")
```

(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')
```

تاريخ / /

موضوع:

(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=
                                  self.remove_task)
        remove_button.pack()
    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, columnspan=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, padx=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()
        |
root = tk.Tk()
app = CalculatorApp(root)
root.mainloop()
```

تاریخ: / /

موضوع:

(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()
```

⑩ 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)
        except: print("not found")
root = tk.Tk()
app = WeatherApp(root)
root.mainloop()
```