1- create a class called person with athributes 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("John", 30)
print(f"Name: {person1.name}, Age: {personal.age}")
```

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

```
classperson:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def greet(self):
        print(f"Hello, {self.name}!")
Person1 = Person("John", 30)
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"Make: {self.make}, Model: {self.model}, year: {self.year}")

car1 = car ("Toyota", "camry", 2020)
car1. car = details()

4): creat a class circle with a method to compute the area. Initialize the class with radius.

import math
class circle:
def _ init _ (self, radius):
self. radius = radius
def compute_area(self):
area = math. pi * (self. radius ** 2)
return area
circle1 = circle(5)
print(f"Area of the circle: {circle1.area()}")

5): create class rectangle with method to compute the area and perimeter. initailize the class with the length and width.

```
class rectangle:
  def __init__(self, length, width):
    self.length = length
    self.width = width
  def compute_area(self):
    area = self.length * self.width
    return area
  def compute_perimeter(self):
    perimeter = 2 * (self.length + self.width)
    return perimeter
rectangle1 = Rectangle(5,10)
print(f"Area of the rectangle: {rectangle1.compute_area()}")
print(f"Perimeter of the rectangle: {rectangle1.compute_perimeter()}")
```

6): create a base class Animal with a method speak. Creat two derived classes Dog and cat that override the speak method.

```
class Animal:
    def speak(self):
        pass
class Dog (Animal):
    def speak(self):
        return "Bark"
class cat (Animal):
    def speak(self):
        return "Meow"
```

7): create a base class shape with attributes name and salary. Create a derived class Manager with an additional attribute department.

```
class shape:
    def area(self):
        pass
class square (shape):
    def area(self, side_length):
        return side_length * side_length
```

```
class Traingle(shape):
  def area (self, base, height):
    return 0.5 * base * height
```

8): create a class Employee with attribute name and salary. creat derived classes Bike and Truck that override the drive method.

```
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.
```

9): create a base vehicle with method drive. create derived classes Bike and Truck that override the drive method.

```
class vehicle:
    def drive(self):
        Pass
class Bike(vehicle):
    def drive(self):
    return "Riding the bike"
class Truck(vehicle):
    def drive(self):
    return "Driving the truck"
```

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):
    return "Soaring high in the Sky"
class ~~penguin~~ Eagle(Bird):
    def Fly(self):
```

```
class penguin (Bird):
  def Fly (self):
    return "penguins cannot Fly!"
```

11): create a class account with private attributes balances. provide public methods to deposit and withdraw money.

```
class account:
  def_init_(self):
    self_balance = 0
def deposit (self,= amount):
self_balance + = amount
def withdraw (self.amount)
  if amount <= self._balance:
  self_balance_ = amount
    else:
    print (" Insufication Funds ")
```

12): create a class book with private attributes balance. provide public method to ~~deposite~~ get and set these attributes.

```
class book:
  def __init__(self, title, author, pages):
    self.title = title
    self.author = author
    self.page = pages
  def get_title(self):
    return self._title
  def set title(self, title):
    self._title = title
  def get_author(self):
    return self._author
  def set_pages(self.pages):
    self.pages = pages
```

تاريخ: / /

موضوع:

13): Create a class laptop with privide attribute brand, model, and price. Provide a method to apply a discount and method to display laptop details

14): ~~Create a class B~~

```
class laptop:
    def __init__(self, brand, model, price):
        self.brand = brand
        self.model = model
        self.price = price
    def apply discount(self, discount):
        self.price = (self.price * discount/10
    def display_detail(self):
print(f"Brand: {self.brand}, Model:
{self.model}, price: $ {self.price}")
```

14):

```
class bank account:
def __init__(self, account_number, balance):
self._account_number = account_number
self._balance = balance
def deposite(self, amount):
self._balance += amount
def withdraw(self, amount):
if amount <= self._balance:
    self._balance -= amount
else:
    print("Insufficient Funds")
def check_balance(self):
    return self._balance
```

(5):

```
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 set_name(self, name):
    self._name = name
def get_grade(self):
    return self._grade
def set_grade(self, grade)
    self._grade = grade
def get_grade(self):
    return self._grade
def set_grade(self, grade):
    self._grade = grade
def get_age(self):
    return self._age
def display_details(self):
print(f"Name: {self._name}, Grand:
{self._grade}, Age: {self._age}")
```

```
class library:
def __init__ (self, name):
    self.name = name
        self.books = []
def add_book (self, book):
 self.books.append (book)
 def remove book (self, title):
   For book in self.books:
       if book.title == title:
 self.books.remove (book)
       break
```

17):

```
class school:                    self.name = name
def __init__ (self, student): self.students = Student
 self.students.append (student)
def remove-student (self, student_id):
 For student in self.students:
 if students.student_id == student_id:
  self.students.remove (student)
       break
```

~~python~~

```
def __init__(self, name):
    self.name = name
    self.members = []
def add_member(self, member):
    self.member.append(member)
def remove_member(self, member_id):
    for member in self.members:
        if member.member_id == member_id:
            self.members.remove(member)
            break
```

```
class campany:
 def_init_(self, name)
   self name = name
    self.employees = [ ]
def add-employees append (employee)
     break
```

20):

```
    class zoo:
  def_init_(self. name):
      self name =name
      self. animals = [ ]
 def add animal(self, animal):
      self. animals. append (animal)
 def remove_animal (self, animal_id):
   For animal in self. animals:
 if animal in self. animals: id = animal
-id:
  self. animals. remove(animal)
 def remove_animal (self. animal_id):
 For animal in self. animals:
  if animal. animal_id == animal_id:
  self. animals. remove (animal)
    break
```

25):

```
class item:
  def __init__(self, name, price):
      self.name = name
      self.price = price
class shopping cart:
    def __init__(self):
        self.items = []
  def remove_item(self):
      For item in self.items:
print(f" item: {item.name}, price: {item.pri
```

26):

Print (F" movie: {self.movie_name}, Seat Number: {self.seat_number}, Price: {self.price}")

def apply_discount(self, discount)

self.price -= (self.price * discount)

28):

class restaurant:

def __init__(self, name):

self.name = name

self.name = []

def add item_to_menu (self, item):

self.menu.append (item)

def remove_item_from_menu (self, item):

self.menu.remove (item)

29):

```
class person:
 def_init_(self name):
    self.name=name
classflight:
   def_init_(self, Flight-number, destination)
    self.Flight-number=Flight-number
    self.destination= destination
     self.passengers.append(passenger)
def remove-passenger(self, passenger):
    self.passengers.remove(passenger):
```

30):

```
  classRoom:
    def_init_(self, room_number):
      self.room_number=room_number
       self.is_occupied=False
class Hotel:
   def_init_(self, name):
     self.name=name
      self.room=[]
```

```
def book_room (self, room_number):
  For room in self.rooms:
 if room.room_number == room_number:
    room.is_occupied = True
def check_out.room(self, room_number):
     For room in self.rooms:
     if room.room_number == room_number
   room.is_occupied = False
```

36): import Tkinter as tk

```
class counterApp:
  def_init_(self,root):
     self.count=0
     self.lebal= tk.lebal(root, text= str (self))
     self.lebal.pack()
     increment_button = Tk.Button(root, text =
"Increment", command = self.increment)
 increment_button.pack()
  def increment(self):
     self.count += 1
     self Label.config (text = str(self.count))
  def decrement(self):
     self.count += 1
self.Label.config (text= str(self.count))
```

```
2): import tkinter as tk
From tkinter import messagebox
class ToDoApp:
def init(self.master):
    self.master = master
    self.master.title("To Do list")
    self.Frame = tk.Frame(self.master)
    self.Frame.pack(padx=10, pady=10)
    self.task_listbox = tk.Listbox(self.Frame
, width = 50, height=10)
    self.task_listbox.pack(side=tk.LEFT)
    self.scrollbar = tk.Scrollbar(self.Frame)
    self.scrollbar.pack(side=tk.RIGHT, Fill=tk.Y)
    self.task_entry = tk.Entry(self.master,width
=52)
    self.task_entry.pack(pady=10)
    self.add_task_button = tk.button(self.master
, text="addtask", command=self.add_task)
    self.add_task_button.pack(pady=5)
def add_task(self):
    task = self.task_entry.get()
    if task != "":
    self.task_listbox.insert(tk.END,task)
```

```
else:
    messagbox.showwaring("warning", "you must
enter a task")
def remove_task(self):
    try:
    selected_task_index= self.tasks_listbox
    .curselection()[0] self.task_listbox.listbox
    .delete(selected_task_index)
    except IndexError:
    messagbox.showwaring("warning", you
must select a task to remove.")
if __name__ == "__main__":
    root = tk.Tk
    app = ToDoApp(root)
    root.mainloop()
```

38) import tkinter as tk

class calculatorApp:

def __init__(self, root):

self.root = root

self.root.title("simple calculator")

self.result_var = tk.StringVar()

self.result_display = tk.Entry(self.root, textvariable=self.result_var, font=("Arial", 24), justify='right')

self.result_display.grid(row=0, column=0, columnspan=4, padx=10, pady=10)

buttons = [

'7', '8', '9', '/',

'4', '5', '6', '*',

'1', '2', '3', '-',

'C', '0', '=', '+'

row_val = 1

col_val = 0

for button in buttons:

if button in buttons:

if button == '=':

def append_to_expression(self, value):

current_expression = self.result_var.get()

39)

```
import tkinter as tk
From tkinter import messagbox
class LoginApp:
 def__init__(self, master):
  self.master = master
  self.master.title("login Form")
  self.Frame = tk.Frame(self.master)
  self.Frame.pack(padx=10, pady=10)
  self.username_label = tk.label(self.Frame, text
=username:")
  self.username_label.grid(row=0, column=0
padx=5, pady=5)
  self.username_entry = tk.Entry(self.Frame)
  self.username_entry.grid(row=0, column=1
  padx=5, pady=5)
  self.password_label = tk.label(self.Frame,
  text="Password:")
  self.password_label.grid(row=1 column=0
  padx=5, pady=5)

  self.login_button = tk.Button(self.Frame,
text="login" command=self.login)
  self.login_button.grid(row=2, columnspan=2
, pady=10
```

40): import tkinter as tk
import requests
class WeatherApp:
def __init__(self, root):
self.root = root
self.root.tittle("Weather inFormation")
self.api_key = "your actual-api-key"
self.cities = ["Kabul", "Herat", "Mazar", "kandahar"]
self.city_label = tk.label(root, text="Slect a city:")
self.city_label.pack(Pady = 5)
self.city_var = tk.StringVar(value = self.cities[0])
self.city_meny = tk.optionMenu(root, self.city_var, * self.cities)
self.city_menu.pack(Pady=5)
self.Fetch_button = tk.Bouton(root, text="Get weather", command=self.get_weather)
self.Fetch_button.pack(Pady=10)
self.result_text = tk.text(root, height=10 width=50)
self.result_text.pack(Pady=5)

21):

```
class File Manager:
def -init-(SelF. File_path):
  SelF. File_path = File_path
def read_File(selF):
"""  read the content OF the File"""
try:
  with open(SelF. File_path, 'r') as
    return File. read()
    except File Not Found Error
    return 'File not Found'
    except Exception as e:
      return str(e)
  def write_File (self, content):
  with open(SelF. File_path, 'w') as
  File:
    File. write (content)
    return 'write success FuL'
    except Exception as e:
      return str(e)
  File_manager =
    File Manager('exampl. txt')
print (File_manager. read File())
```

22): ~~create a class log~~

```
class log:
def __init__(self,
File_name='myFile.txt):
    self.File=name=File_name
    def write_error(self, message):
with open(self.File_name, 'a') as
log-File:
timeStamp=
date time.datetime.now().strftime('%Y
-%m-%d  %H:%M:%S')
    log-File.write(F'{timestamp}
ERROR:{message}\n')
log=log()
log.write-error('this is an errormessa
```

23):

```
class config
def __init__(self,
Filename='config.ini'):
self.config =
configparser.configparser()
self.config.read(Filename)
def get_setting(self, section, option):
try:
return self.config.get(section, option
except
(configparser.NoSectionError,
configparser.NoOptionError as e:
print(f'Error:{e}')
return NONE
def get_int_setting(self, section, option):
try:
return
self.config.getint(section, option)except)
except
(configparser.NoSectionError, configparser
.NooptionError, valueError) ase:
print('Error: {e}')
return None
```