

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("John", 30)
```

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

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(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) Create 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)

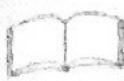
return area

(13")

circle1 = circle(5)

- area

print(f"Area of the circle: {circle1.compute_area():.2f} square units")



5): create class rectangle with method
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 compute_area(self):

area = self.length * self.width

return area

def compute_perimeter(self):

perimeter = 2 * (self.length + self.width)

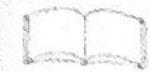
return perimeter

rectangle1 = Rectangle(5, 10)

print("Area of the rectangle: {}")
rectangle1.compute_area()

print("Perimeter of the rectangle:

{})



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):

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



1 / 10

Topic 1

class Triangle(shape):

def area(self, base, height):

return 0.5 * base * height

8) create a class Employee with attribute name and salary. Create 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.

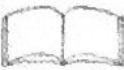
class Bird:

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!"

ii): create a class account with private attributes balance. 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("Insufficient Funds")

12) Create a class book with private attributes balance. provide public method to ^{get} deposit 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):

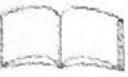
self.title = title

def get_author(self):

return self.author

def set_pages(self, pages):

self.pages = pages



(3): Create a class Laptop with private attribute brand, model, and price. Provide a method to apply a discount and method to display laptop details

(4): Create a class &

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) / 100

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 deposit(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}, Grade: {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

if:

class School:

 self.name = name

def __init__(self, students):

 self.students = students

 self.students.append(student)

def remove_student(self, student_id):

 for student in self.students:

 if 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.members.append(member)

def remove_member(self, member_id):

for member in self.members:

if member.id == member_id:

self.members.remove(member)

break

class company:

def __init__(self, name):

self.name = name

self.employees = []

def add_employees(append(employees))

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.id == 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



27):

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.price}")

28):



Print ("Movies: {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

class flight:

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):

class Room:

def __init__(self, room_number):

self.room_number = room_number

self.is_occupied = False

class Hotel:

def __init__(self, name):

self.name = name

self.rooms = []

```
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.label = tk.Label(root, text=str(self))
```

```
    self.label.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))
```

1. 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)  
        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_tasks_button = tk.Button(self.master,  
                                         text="add task", command=self.add_task)  
        self.add_tasks_button.pack(pady=5)  
    def add_task(self):  
        task = self.task_entry.get()  
        if task != "":  
            self.task_listbox.insert(tk.END, task)
```

else:

 messagebox.showwarning("warning", "you must enter a task")

def remove_task(self):

 try:

 selected_task_index = self.task_listbox

 curselection()[0] self.task_listbox Listbox

 .delete(selected_task_index)

 except IndexError:

 messagebox.showwarning("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', '-',  
            '0', '=', '+']
```

row_val = 1

col_val = 0

for button in buttons:

if button in buttons:

if button == '=':

def append_to_expression(self, value):

convert expression into root node

39) :
import tkinter as tk
from tkinter import messagebox

```
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.title("Weather Information")  
        self.api_key = "your actual api key"  
        self.cities = ["Kabul", "Herat", "Mazar",  
                      "Kandahar"]  
        self.city_label = tk.Label(root, text="Select  
a city:")  
        self.city_label.pack(pady=5)  
        self.city_var = tk.StringVar(value=self  
                                     .cities[0])  
        self.city_menu = tk.OptionMenu(root,  
                                      self.city_var, *self.cities)  
        self.city_menu.pack(pady=5)  
        self.Fetch_button = tk.Button(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 FileManager:

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

file:

return file.read()

except FileNotFoundError:

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 successful'

except Exception as e:

return str(e)

file_manager =

FileManager('example.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 =

datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

- %m - %d %H: %M: %S)

log_file.write(f'{timestamp}\n{message}\n')

ERROR: {message}\n)

log = Log()

log.write_error('this is an error message')

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 (configparser.NoSectionError, configparser.  
                NoOptionError, ValueError) as e:  
            print(f'Error: {e}')  
        return None
```