1. create a class called person with attributes name and age. Create an object of the class prints it attributes

```
class person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
ali = person('ali', 20)
print(ali.name)
print(ali.age)
```

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

```
class person:
    def __init__(self, name):
        self.name = name
    def greet(self):
        return(f'{name} How are you')
ali = person('ali')
print(ali.greet())
```

ils

(self, make, model, year):

make = make

maale = maale

year = year

_self):

its make {self.make} in {self.year}

and its model is {self.model}'

, 2000, 2024)

ls())

ircle with a method to compute

(rectangle.area())

5. create a class rectangle
perimeter. initialize the class

```
class rectangle:
    def __init__(self
        self.length =
        self.width =
    def area(self):
        return f'{s
    def perimeter(sel
        return f'
compute = rectangle(4, 6
print(compute.area())
```

(self):

layuanat'

:

(self):

'Board'

:

-(self):

'meiyou'

```
        self.tool_ar2 = t
    def area(self):
        return self.tool_

class tringle(shape):
    def __init__(se
        self.base=
        self.heigh
    def area(self)
        return 0.5

square = square(4)
print(square.area())
trin = tringle(3,4)
print(trin.area())
```

5- create a base class employee with attributes name and salary. create a derived class manager with additional attributes department.

```
class employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def display_info(self):
        return f'{self.nam} has {self.salary} salary
class manager(employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department
    def display_info(self):
        return f'{self.name} with {self.salary} in
                {self.department} is'
                self.
emp = employee('Ali', 2000)
print(emp.display_info())

mng = manager('Ahmad', 2000, 'S_E')
```

derived classes bike and truck that override the derive method.

```
class vehicle:
    def derive(self):
        print('riding a vehicle')
class bike(vehicle):
    def derive(self):
        print('riding a bike')
class truck(vehicle):
    def derive(self):
        print('riding a bike')

v= vehicle()
v.vehicle.derive()
```

10. create a base class bird with a method fly. create derived classes eagle and penguin override the fly method in penguin to indicate the penguins cannot fly.

```
class bird:
    def fly(self):
        print('some birds can flying')
class eagle(bird):
    def fly(self):
        print('eagle can flying')
class penguin(bird):
    def fly(self):
        print('penguin can not flying')

b=bird()
b.fly()

eag=eagle()
eag.fly()

pen=penguin()
pen.fly()
```

11. create a class account with private attributes balance. provide public methods deposit and withdraw money.

```
class account :
    def __init__(self, balance):
        self.__balance = balance
    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
    def withdraw(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount

acc = account(100)
acount.deposite(500)
acc.withdraw(200)
```

```
        if amount > 0:
            self.__balance += amount
            print(f'{amount} {self.__balance}')
    def withdraw(self, amount):

        if 0 < amount <= self.__balance:
            self.__balance -= amount
            print(f'{amount} {self.__balance}')
        else:
            print('you don't have money')
acc = account(1000)
acc.deposit(500)
acc.withdraw(200)
```

12. ... private attributes, ~~author~~title, author, and pages. provide public methods to get and set this attributes

```
class book:
    def __init__(self, title, author, pages):
        self.__title = title
        self.__author = author
        self.__pages = pages
    def get_title(self):
        return self.__title
    def get_author(self):
        return self.__author
    def get_pages(self):
        return 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
bo = book('1403', 'Ali', 328)
print(b.get_pages(1)
book.set_title('Afghanistan')
bo.set_pages('112)
bo.set_autho('Ali')
```

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
```

14. create a class ... private attributes account_number and balance. provide methods to deposit, withdraw, and check the balance.

```
class BankAccount:
    def __init__(self, account_number, balance):
        self.__account_number = account_number
        self.__balance = balance
    def deposit(self, amount):
        if amount>0:
            self.__balance += amount
            print(f'deposited{amount} Now is{self.__ba
        else:
            print('deposit amount be positive')
    def withdraw(self, amount):
        if 0< amount <= self.__balance:
            self.__balance -= amount
            print{f' with_{amount} Now is {self.__bala
    def check_balance(self):
        return self.__balance
account = BankAccount('8200240', 1000)
account.deposit(50)
account.withdraw(30)
print(account.check_balance())
```

15 create a class student with private attributes name, grade and age. provide methods to get and set this attributes and a method to display the students details.

```
class student:
    def __init__(self, name, grade, age):
        self.__name = name
        self.__grade = grade
        self.__age = age
    #getter methods
    def get_name(self):
        return self.__grade
    def get_grade(self):
        return get_grade
    def get_age(self):
        return get_age
    def set_name(self, name):
        self.__name = name
    def set_grade(self, grade):
        self.__grade = grade
    def set_age(self, age):
        set_age = age
    def display_details(self):
        print(f"{self.__name}")
```

Advanced concepts exercise

16. create a class Library with attributes name and books (a list of book objects). Provide methods to add and remove members books.

```
class book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
    def __str__(self):
        return f'{self.title} by {self.author}'

class library:
    def __init__(self, name):
        self.name = name
        self.books = []
    def add_books(self, book):
        if instance(book, book):
            self.books.append(book)
            print(f'{book} added to library')
        else:
            print('only book objects can be add')
    def removed_book(self, book):
        if book in self.books:
            self.books.remove(book)
            print(f'{book} removed')
```

17. create a class school with attributes name and Student (a list of students objects). provide methods to add and remove ~~members~~ students.

```
def display_books(self):
    if self.books:
        print(f'books in {self.name}:')
        for book in self.books:
            print(f' - {book}')
    else:
        print('No books in library')

library = Library('city library')
book1 = book(1202, 'Ahmad')
book2 = book('Horzon', 'Mahmood')

library.add_book(book1)
library.add_book(book2)
library.display_books()

library.remove_book(book2)
```

15. Create a ... attributes name and Students (a list of student objects). Provide methods to add and remove students.

```
class students:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age
    def get_name(self):
        return self.__name
    def get_age(self):
        return self.__age
    def set_name(self, name):
        self.__name = name
    def set_age(self, age):
        self.__age = age
    def display_details(self):
        print(self.__name)
        print(self.__age)
    def __str__(self):
        return f'{self.name}, {self.age}'

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

```
    def add_students (self, student):
        if instance(student, student):
            self.students.append(student)
            print(f' added {student.get_name()}')
        else:
            print('only student should be added')
    def remove_student(self, student):
        if student in self.students:
            self.students.remove(student)
            print(f' removed {stident.get_name()}')
        else:
            printl'student not found')
    def display-student(self):
        if self.students:
            print(f' students' in {self.name}')
            for student in self.students:
                print(f' - {student}')
        else:
            print(' No students in this school')

school = School('dahe surkh')
st1 = student('John', 15)
st2 = Student('zzz', 17)
```

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, species, name):
        self.species = species
        self.name = name
    def __str__(self):
        return f'{self.name} {self.species}'

class zoo:
    def __init__(self, name):
        self.name = name
        self.animals = []
    def add_animal(self, animal):
        if isinstance(animal, animal):
            self.animals.append(animal)
            print(f'added {animal.name}')
        else:
            print('animal should added')
    def remove_animals(self):
        if self.animals
```

```
def remove_animal(self, animal):
    if animal in self.animals:
        self.animal.remove(animal):
        print(f'{animal.name} removed')
    else:
        print('animal not found')
def display_animals(self):
    if self.animals:
        print(f'animals in {self.name}')
        for animal in self.animals:
            print(f'- {animal}')
    else:
        print('no animals in the zoo')
zoo = Zoo('city zoo')
animal1 = animal('lion', 'Leo')
animal2 = animal('cat', 'demo')
zoo.add_animal(animal1)
zoo.addanimal(animal2)
zoo.display-animals()
zoo.remove_animal(animal1)
zoo.display_animals()
```

```
lf, filename):
ame = filename
ile (self, content):

en(self.filename, 'w') as file
file.write(content)
print(f'wrote {self.filename}.')
xception as e:
f'an error while writing to file: {e}')
e(self):

open(self, name, 'r') as file:
content = file.read()
print(f'read from {self.filename}')
FileNotFoundError:
int(f'the file {self.filenam} not')
eturn None
Exception as e:
int('an error found while reading')
```

to a log file.

```
class log:
    def __init__(self, log_filename):
        self.log_filename = log_filename
    def write_error(self, error_message):
        try:
            with open(self.log_filename, 'a') as log_fi
                log_file.write(f'{error_message}\n')
                print(f'{self.log_filename}')
        except Exception as e:
            print(f'{self.log_filename}')

log = log('error_log.txt')

log.write_error('this is log_error message')
log.write_error('another error occured')
```

23. write a class ... configuration settings from a file and provides methods to access these settings.

```
class config:
import json
class config:
    def __init__(self, config_filename):
        self.config_filename = config_filename
        self.settings = {}
        self.load_config()
    def load_config(self):
        try:
            with open(self.config_filenam,'r'):
            self.settings = json.load(config_file)
            print(f'{self.config_filename}')
    except FileNotFoundError:
        print(f'{self.config_filename} not exist')
    except json.JSONDecodeError:
        print(f'error decoding {self.config_filename}')
    except Exception as e:
        print(f'an error: {e}')
```

```
def get_setting(self, key):
        return self.settings.get(key, None)
config = Config('config.json')
database_url = config.get_setting('database_url')
if database_url:
    print(database_url)
else:
    print('database_url not found')
```

24. create a datase that connects to a database and provid methods to execute queries handle exceptions if the connecti fails.

تاريخ: / /

موضوع:

Q- Creat a class flight with the attribute flight number, destination and Passenger (a list of Passenger) Provid a method to add and remove Passenger

```
class Passenger:
    def __init__(self, flight_number,
class Passenger:
    def __init__(self,name, age):
        self.name=name
        self.age=age
    def __str__(self):
        return {self.name}, {self.age}
class flight(Passenger):
    def __init__(self, flight_number), distination):
        self.flight_number = flight_number
        self.distination=distination
        self.Passengers=[]
    def add_Passenger(self,name):
        self.nam
        self.Passenger.append(name)
    def remove_Passenger(self, name):
        if name in self.Passengers:
            self.Passenger.remove(naw)
obj = flight(134,"Canada")
obj.add_Passenge("Shoaib")
```

```
import tkinter as tk
class CounterAPP:
    def __init__(self,root):
        self.root = root
        self.root.title("Counter Application"
        self.counter = 0
        self.label = tk.Label(root, text=str(se
        font=("Arial",18)))
        self.label.pack(pady=10)
        self.increment_button = tk.Button(root,
        text="increment", command=self.increme
        self.increment.pack(side=tk.LEFT,padx
        self.decrement_button = Button(root,
        text="decement", command=self.decremen
```

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

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

```
tk.
er as tk
r import *
)p.

etf
.init_(self, root)
self.root = root
self.root.title("To Do list APP
self.task_entry = tk.Entry(root, width=
self.task_entry.pack(pady=10)
self.add_button = tk.Button(root, text="
command = self.add_task
```

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