Frédéric
LURET

Python
Banque de questions

Version du 15 décembre 2024

Université de TOURS

0 Table des matières

1 Site 8 Class

Dictionnary de comprehension

Question 1

DEPART de la classe Ecrire un programme python pour créer une classe d'étudiants

Code python :

```python
class Student():
    id = 0
    name = ""
    gender = ""
    total = ""
    per = 0

    def setData(self,id,name,gender,total,per):  # function to set data
        self.id = id
        self.name = name
        self.gender = gender
        self.total = total
        self.per = per

    def showData(self):  # function to get/print data
        print("Id :",self.id)
        print("Name :", self.name)
        print("Gender :", self.gender)
        print("Total :", self.total)
        print("Percentage :", self.per)

s = Student()
s.setData(1,'Sam Kumar','Male',422,84.44)
s.showData()
```

q710.py

Ce programme Python définit une classe Student avec des attributs et des méthodes permettant de définir et d'afficher les données des étudiants. Voici une explication du programme :

classe Étudiant : Il s'agit de la classe principale représentant un étudiant. Elle possède les attributs et méthodes suivants : Attributs de la classe : id : Permet de stocker

l'identifiant de l'étudiant (initialisé à 0 par défaut). name : Stocke le nom de l'étudiant (initialisé par défaut à une chaîne vide). gender (sexe) : Indique le sexe de l'étudiant (initialisé par défaut comme une chaîne vide). total : Stocke les notes totales de l'étudiant (initialisé par défaut comme une chaîne vide). per : Stocke le pourcentage de l'étudiant (initialisé à 0 par défaut). setData(self, id, name, gender, total, per) : Cette méthode est utilisée pour définir les données d'un étudiant. Elle prend en paramètre l'identifiant, le nom, le sexe, le total des notes et le pourcentage de l'élève et les affecte aux attributs correspondants de l'instance. showData(self) : Cette méthode est utilisée pour afficher les données de l'étudiant. Elle affiche l'ID, le nom, le sexe, le total des notes et le pourcentage de l'étudiant. Une instance de la classe Student est créée et nommée s. La méthode setData est appelée sur l'instance s pour définir les données de l'étudiant. Dans ce cas, l'ID de l'étudiant est fixé à 1, son nom à "Sam Kumar", son sexe à "Male", le total de ses notes à 422 et son pourcentage à 84,44. La méthode showData est appelée sur l'instance s pour imprimer les données de l'étudiant.

Question 2

Écrire un programme python pour créer une classe d'étudiants avec un constructeur et un destructeur.

Code python :

```python
class Student:
    def __init__(self): #Constructor
        self.id = 0
        self.name = ""
        self.gender = ""
        self.total = ""
        self.per = 0

    def __del__(self): #Destructor
        print("Object Destroyed")

    def setData(self):
        self.id = int(input("Enter a ID :"))
        self.name = input("Enter a Name :")
        self.gender = input("Enter a Gender :")
        self.total = int(input("Enter a Total :"))
        self.per = float(input("Enter a Percentage :"))

    def showData(self):
        print("Id :",self.id)
        print("Name :", self.name)
        print("Gender :", self.gender)
        print("City :", self.total)
        print("Salary :", self.per)

s = Student()# Create an instance of the Student class
s.setData()# Set data for the instance
s.showData()# Display data for the instance
```

q711.py

Ce code Python définit une classe Student avec un constructeur, un destructeur et des méthodes pour définir et afficher les données des étudiants. Voici une explication du code :

class Student : : Cette ligne définit une classe nommée Student, qui sert de modèle pour la création d'objets étudiants. def __init__(self) : : Il s'agit de la méthode de construction de la classe Étudiant. Le constructeur est automatiquement appelé lorsqu'une instance de la classe est créée. À l'intérieur du constructeur : Des valeurs par défaut sont attribuées aux variables d'instance (attributs) telles que id, name, gender, total et per. Ces attributs représentent l'identifiant, le nom, le sexe, le total des notes et le pourcentage de l'étudiant. def __del__(self) : : Il s'agit de la méthode de destruction de la classe Student. Le destructeur est appelé lorsqu'un objet est sur le point d'être détruit. Dans ce cas, il affiche simplement "Object Destroyed" lorsque l'objet est détruit. def setData(self) : : Cette méthode permet de définir les données d'une instance d'étudiant. Elle demande à l'utilisateur de saisir des données et définit les valeurs des attributs de l'élève en fonction des données saisies par l'utilisateur. def showData(self) : : Cette méthode affiche les données de l'élève. Elle imprime les valeurs des attributs de l'étudiant dans la console. s = Student() : Cette ligne crée une instance de la classe Student et l'affecte à la variable s. Cette instance représente

un étudiant spécifique. s.setData() : La méthode setData est appelée sur l'instance s pour définir les données de cet étudiant. L'utilisateur est invité à saisir l'identifiant, le nom, le sexe, le total des notes et le pourcentage de l'élève. s.showData() : La méthode showData est appelée sur l'instance s pour afficher les données de l'élève. Elle imprime les valeurs des attributs définis dans la méthode setData.

Lorsque vous exécutez ce programme, il crée un objet Étudiant, définit ses attributs en fonction des données saisies par l'utilisateur, puis affiche les informations relatives à l'étudiant.

Question 3

Ecrire un programme python pour implémenter des Getters et des Setters dans une classe.

Code python :

```python
class Student:
    def __init__(self): #Constructor
        self.name = ""
        self.total = ""
        self.per = 0

    def setName(self,name):
        self.name = name
    def getName(self):
        return self.name

    def setTotal(self,total):
        self.total = total
    def getTotal(self):
        return self.total

    def setPercentage(self,per):
        self.per = per
    def getPercentage(self):
        return self.per

# Get user input for name, total, and percentage
name = input("Enter a Name :")
total = int(input("Enter a Total :"))
per = float(input("Enter a Percentage :"))

s = Student()

#Set the attributes using setter methods
s.setName(name)
s.setTotal(total)
s.setPercentage(per)

# Get the attributes using getter methods
n = s.getName()
t = s.getTotal()
p = s.getPercentage()

print("\nDisplaying Student Details")
print("Name :", n)
print("Total :", t)
print("Percentage :", p)
```

q712.py

Ce programme Python définit une classe d'étudiant qui encapsule les détails de l'étudiant tels que le nom, le total et le pourcentage. Il démontre l'utilisation des méthodes setter et getter pour définir et récupérer ces attributs. Voici une explication du programme :

classe Étudiant : : Il s'agit de la définition de la classe Student. def __init__(self) : :

Il s'agit de la méthode de construction de la classe Étudiant. Elle initialise trois variables d'instance : name, total et per avec des valeurs par défaut. self.name = "", self.total = "", self.per = 0 : Ces lignes initialisent le nom comme une chaîne vide, le total comme une chaîne vide et per comme 0. def setName(self, name) : : Il s'agit d'une méthode setter permettant de définir le nom de l'élève. self.name = name : L'attribut name est mis à jour avec la valeur passée à la méthode setName. def getName(self) : : Il s'agit d'une méthode getter permettant de récupérer le nom de l'élève. return self.name : L'attribut name est renvoyé lorsque la méthode getName est appelée. Des méthodes setter et getter similaires sont définies pour les attributs total et par. name = input("Enter a Name :") : Le programme invite l'utilisateur à saisir un nom et la saisie est stockée dans la variable name. total = int(input("Enter a Total :")) : Le programme invite l'utilisateur à saisir un total, et la saisie est convertie en un nombre entier et stockée dans la variable total. per = float(input("Enter a Percentage :")) : Le programme invite l'utilisateur à saisir un pourcentage, et la donnée est convertie en un nombre flottant et stockée dans la variable per. s = Student() : Une instance de la classe Student est créée, représentant un étudiant. s.setName(name), s.setTotal(total), s.setPercentage(per) : Les méthodes de définition sont utilisées pour définir le nom, le total et le pourcentage de l'étudiant en fonction des données fournies par l'utilisateur. n = s.getName(), t = s.getTotal(), p = s.getPercentage() : Les méthodes Getter sont utilisées pour récupérer le nom, le total et le pourcentage de l'élève. print("
nAffichage des détails de l'étudiant") : Un message est imprimé pour indiquer que les détails de l'élève vont être affichés. print("Nom :", n), print("Total :", t), print("Pourcentage :", p) : Le nom, le total et le pourcentage de l'étudiant sont affichés sur la console en utilisant les valeurs récupérées par les méthodes getter.

Ce programme démontre l'utilisation des principes orientés objet, de l'encapsulation et des méthodes setter et getter pour gérer et récupérer les informations relatives aux étudiants. Il permet à l'utilisateur de saisir des informations sur l'étudiant et affiche ensuite les détails de l'étudiant.

Question 4

Écrire un programme python pour mettre en œuvre l'abstraction à l'aide d'une classe abstraite.

Code python :

```python
#Abstract Class
class Vehicle:
    def acclerate(self,name):
        pass
    def park(self,name):
        pass

class Bike(Vehicle):
    def acclerate(self, name):
        print(name,"is accelrating @ 60kmph")
    def park(self, name):
        print(name,"is parked at two wheeler parking")

class Car(Vehicle):
    def acclerate(self, name):
        print(name,"is accelrating @ 90kmph")
    def park(self, name):
        print(name,"is parked at four wheeler parking")


c = Car()
c.acclerate("Car")
c.park("Car")

b=Bike()
b.acclerate("Bike")
b.park("Bike")
```

q713.py

Ce programme Python démontre l'utilisation de classes abstraites et de la superposition de méthodes dans la programmation orientée objet. Il définit une classe de base abstraite, Vehicle, et deux sous-classes concrètes, Bike et Car. La classe Vehicle contient deux méthodes abstraites, accelerate et park, qui sont surchargées par les classes enfants afin de fournir des implémentations spécifiques pour différents types de véhicules. Voici une explication du programme :

classe Véhicule : : Il s'agit de la classe de base abstraite Véhicule. Elle contient deux méthodes abstraites, accelerate et park, qui sont définies mais n'ont pas d'implémentation (indiquée par l'instruction pass). Les méthodes abstraites sont des méthodes qui sont déclarées dans la classe de base mais qui n'ont pas d'implémentation spécifique, et elles doivent être surchargées par les classes enfants. classe Vélo(Véhicule) : : Il s'agit de la classe Bike, qui hérite de la classe Vehicle. Elle surcharge les méthodes accelerate et park pour fournir un comportement spécifique à un vélo. classe Car(Véhicule) : : Voici la classe Voiture, qui hérite également de la classe Véhicule. Elle surcharge les méthodes d'accélération et de stationnement pour fournir un comportement spécifique à une voiture. def accelerate(self, name) : : Dans les classes Bike et Car, la méthode accelerate est implémentée pour accepter un paramètre name et imprimer un message spécifique au type de véhicule, indiquant qu'il accélère. def park(self, name) : : Dans les classes Bike et Car, la méthode park est implémen-

tée pour accepter un paramètre name et imprimer un message spécifique au type de véhicule, indiquant qu'il est en train de se garer. c = Voiture() : Une instance de la classe Voiture est créée et affectée à la variable c. c.accelerate("Voiture") : La méthode accelerate est appelée sur l'objet c, en passant "Car" comme paramètre de nom. Elle imprime un message indiquant que la voiture accélère. c.park("Voiture") : La méthode park est appelée sur l'objet c, en passant "Voiture" comme paramètre de nom. Elle imprime un message indiquant que la voiture est en train de se garer. b = Bike() : Une instance de la classe Bike est créée et assignée à la variable b. b.accelerate("Vélo") : La méthode accelerate est appelée sur l'objet b, en passant "Bike" comme paramètre de nom. Elle imprime un message indiquant que le vélo accélère. b.park("Vélo") : La méthode park est appelée sur l'objet b, en passant "Bike" comme paramètre de nom. Elle imprime un message indiquant que le vélo est en train de se garer.

Dans ce programme, l'utilisation d'une classe de base abstraite (Véhicule) garantit que toutes les sous-classes (Vélo et Voiture) fournissent des implémentations spécifiques pour les méthodes accelerate et park. Cela vous permet de créer différentes instances de véhicules (vélo et voiture) et d'appeler leurs méthodes spécifiques pour démontrer le comportement de chaque type de véhicule.

Question 5

Écrire un programme python pour implémenter une interface à l'aide d'une classe

Code python :

```python
import math
class Shape: #Interface
    def input(self):pass
    def process(self):pass
    def output(self):pass

class Circle(Shape):
    def __init__(self,rad=0.0):
        self.radius = rad
        self.area = 0.0

    def setdata(self):
        self.radius = float(input("Enter radius :"))

    def circle_area(self):
        self.area = math.pi*math.pow(self.radius,2)

    def getdata(self):
        print("Circle Area :",self.area)

class Rectangle(Shape):
    def __init__(self,len=0,bre=0):
        self.len = len
        self.bre = bre
        self.area = 0

    def setdata(self):
        self.len = int(input("Enter Length :"))
        self.bre = int(input("Enter Breadth :"))

    def rect_area(self):
        self.area = self.len*self.bre

    def getdata(self):
        print("Rectangle Area :",self.area)

c = Circle()
c.setdata()
c.circle_area()
c.getdata()

r = Rectangle()
r.setdata()
r.rect_area()
r.getdata()
```

q714.py

Ce programme Python définit deux classes, Circle et Rectangle, qui héritent d'une classe de base commune, Shape. Le programme calcule les surfaces des cercles et des

rectangles en fonction des données saisies par l'utilisateur. Voici une explication du programme :

Importation du module mathématique : Le programme commence par importer le module mathématique pour accéder à des fonctions mathématiques telles que pi et pow. Classe Shape : Il s'agit de la classe de base, qui peut être considérée comme une interface dans ce contexte. Elle définit trois méthodes : input, process et output. Ces méthodes sont des espaces réservés pour des implémentations spécifiques dans les classes dérivées, mais ne sont pas implémentées ici. classe Circle(Shape) : Il s'agit de la classe Circle, qui hérite de la classe Shape. Elle représente un cercle et est responsable du calcul de sa surface. __init__(self, rad=0.0) : Le constructeur initialise l'objet Circle avec un rayon par défaut de 0,0. Il initialise également une variable de surface à 0,0. setdata(self) : Cette méthode invite l'utilisateur à saisir le rayon du cercle et le fixe dans l'attribut radius. circle_area(self) : Cette méthode calcule la surface du cercle à l'aide de la formule $\pi * radius^2$ et stocke le résultat dans l'attribut area. getdata(self) : Cette méthode imprime la surface calculée sur la console. classe Rectangle(Forme) : Il s'agit de la classe Rectangle, qui hérite de la classe Shape. Elle représente un rectangle et calcule sa surface. __init__(self, len=0, bre=0) : Le constructeur initialise l'objet Rectangle avec des valeurs de longueur et de largeur par défaut de 0. Il initialise également un attribut de surface à 0. setdata(self) : Cette méthode invite l'utilisateur à saisir la longueur et la largeur du rectangle et les définit dans les attributs respectifs. rect_area(self) : Cette méthode calcule la surface du rectangle en multipliant la longueur et la largeur et stocke le résultat dans l'attribut area. getdata(self) : Cette méthode imprime la surface calculée sur la console. Création d'instances de cercle et de rectangle : c = Circle() : Crée une instance de la classe Circle. c.setdata() : Invite l'utilisateur à saisir le rayon et le fixe. c.circle_area() : Calcule la surface du cercle. c.getdata() : Affiche la surface du cercle. r = Rectangle() : Crée une instance de la classe Rectangle. r.setdata() : Invite l'utilisateur à saisir la longueur et la largeur et les fixe. r.rect_area() : Calcule la surface du rectangle. r.getdata() : Affiche la surface du rectangle.

En résumé, ce programme montre comment les classes peuvent hériter d'une classe de base commune (ou d'une interface) et comment chaque classe dérivée fournit ses propres implémentations pour des méthodes spécifiques. Il calcule et affiche les surfaces des cercles et des rectangles en fonction des entrées de l'utilisateur.

Question 6

Écrire un programme pour l'héritage simple en Python

Code python :

```python
class Details:
    def __init__(self):
        self.name = ""
        self.total = 0
        self.per = 0

    def setData(self,name,total,per):
        self.name = name
        self.total = total
        self.per = per

    def showData(self):
        print("Name :", self.name)
        print("Total :", self.total)
        print("Percentage :", self.per)

class Student(Details): #Inheritance
    def __init__(self):
        self.Grade = ""

    def setStudent(self,name,total,per,grade):
        self.setData(name,total,per)
        self.grade = grade

    def showStudent(self):
        self.showData()
        print("Grade :", self.grade)


e=Student()
e.setStudent("Kim",430,86.00,'B')
e.showStudent()
```

q715.py

Ce programme Python démontre l'héritage dans la programmation orientée objet. Il définit deux classes, Détails et Étudiant, où la classe Étudiant hérite de la classe Détails. Voici une explication du programme :

classe Détails : : Il s'agit de la classe Détails, qui sert de classe mère. __init__(self) : La méthode du constructeur initialise trois variables d'instance : name, total et per. Toutes ces variables sont initialement définies à 0 ou à une chaîne vide. setData(self, name, total, per) : Cette méthode permet de définir les valeurs des variables d'instance name, total et per. showData(self) : Cette méthode affiche les valeurs de name, total et per. class Student(Details) : : La classe Student est une classe enfant qui hérite de la classe Details. Cela signifie qu'elle hérite des attributs et des méthodes de la classe mère. __init__(self) : La méthode du constructeur de la classe Student initialise une variable d'instance supplémentaire, Grade, qui prend initialement la forme d'une chaîne vide. setStudent(self, name, total, per, grade) : Cette méthode permet de définir les valeurs de name, total, per et de la nouvelle variable d'instance grade.

Elle appelle la méthode setData de la classe mère pour définir les attributs communs. showStudent(self) : Cette méthode appelle d'abord la méthode showData de la classe mère pour afficher les détails hérités de Details. Elle affiche ensuite l'attribut grade. e = Student() : Une instance de la classe Student est créée et assignée à la variable e. e.setStudent("Kim", 430, 86.00, 'B') : La méthode setStudent est appelée sur l'objet e pour définir les détails de l'étudiant, y compris le nom, le total, le pourcentage et la note. e.showStudent() : La méthode showStudent est appelée pour afficher les détails de l'étudiant, notamment son nom, son total, son pourcentage et sa note.

Lorsque vous exécutez le programme, il crée une instance de la classe Student, définit les détails de l'étudiant à l'aide de la méthode setStudent, puis affiche ces détails à l'aide de la méthode showStudent. Ce programme illustre le concept d'héritage, dans lequel la classe Student hérite des attributs et des méthodes de la classe Details et les étend avec ses propres attributs et méthodes.

Question 7

Écrire un programme d'héritage avec deux classes enfantines (dérivées) en Python

Code python :

```python
class Details:
    def __init__(self):
        self.idn = 0
        self.name = ""
        self.gender = ""

    def setDetails(self):
        self.idn = input("Enter the ID Number : ")
        self.name = input("Enter the Name : ")
        self.gender = input("Enter the Gender : ")

    def showDetails(self):
        print("ID : ",self.idn)
        print("Name : ",self.name)
        print("Gender : ",self.gender)

class Student(Details):
    def __init__(self):
        self.total = 0
        self.per = 0

    def setStudent(self):
        self.setDetails()
        self.total = int(input("Enter the Total Mark : "))
        self.per = float(input("Enter the Percentage : "))

    def showStudent(self):
        self.showDetails()
        print("Total : ",self.total)
        print("Percentage : ",self.per)

class Staff(Details):
    def __init__(self):
        self.depart = ""
        self.salary = ""

    def setStaff(self):
        self.setDetails()
        self.depart = input("Enter the Department : ")
        self.salary = float(input("Enter the Salary : "))

    def showStaff(self):
        self.showDetails()
        print("Department : ",self.depart)
        print("Salary : ",self.salary)

print("Student Details : ")
stu = Student()
stu.setStudent()
stu.showStudent()

print("\nStaff Details : ")
stf = Staff()
stf.setStaff()
stf.showStaff()
```

Question 8

Écrire un programme pour l'héritage multiple en Python

Code python :

```python
class PersonalInfo:  # parent classe
    def __init__(self, idn, name, gender, address, contact):
        self.idn = idn
        self.name = name
        self.gender = gender
        self.address = address
        self.contact = contact

    def display_personal_info(self):
        print("Id :   ", self.idn)
        print("Name : ", self.name)
        print("Gender : ", self.gender)
        print("Address : ", self.address)
        print("Contact : ", self.contact)

class AcademicInfo:  # parent classe
    def __init__(self, stream, year):
        self.stream = stream
        self.year = year

    def display_academic_info(self):
        print("Stream : ", self.stream)
        print("Year :   ", self.year)

class Student(PersonalInfo, AcademicInfo):  #child class inheriting from both parent classes
    def __init__(self, idn, name, gender, address, contact, stream, year):
        # Call constructors of parent classes
        PersonalInfo.__init__(self, idn, name, gender, address, contact)
        AcademicInfo.__init__(self, stream, year)

    def display_student_details(self):
        self.display_personal_info()
        self.display_academic_info()

idn = input("Enter the ID : ")
name = input("Enter the Name : ")
gender = input("Enter the Gender : ")
address = input("Enter the Address : ")
contact = input("Enter the Contact : ")
stream = input("Enter the Stream : ")
year = input("Enter the Year : ")

# Create a Student instance
student = Student(idn, name, gender, address, contact, stream, year)

# Display student details
student.display_student_details()
```

q717.py

Ce programme Python démontre le concept d'héritage multiple en créant une classe enfant, Student, qui hérite de deux classes parents, PersonalInfo et AcademicInfo. Le programme recueille les informations personnelles et académiques de l'utilisateur et affiche ensuite les détails combinés de l'étudiant, y compris les informations person-

Question 9

Écrire un programme python pour vérifier les nombres premiers en utilisant une approche orientée objet.

Code python :

```python
class PrimeChecker:
    def __init__(self, num):
        self.num = num

    def is_prime(self):
        if self.num <= 1:
            return False
        if self.num == 2:
            return True
        if self.num % 2 == 0:
            return False

        # Check for divisibility from 3 to the square root of the
        ↪ number
        for i in range(3, int(self.num**0.5) + 1, 2):
            if self.num % i == 0:
                return False

        return True

num = int(input("Enter a Number : "))

checker = PrimeChecker(num)# Create an instance of PrimeChecker

if checker.is_prime():
    print(f"{num} is a Prime Number")
else:
    print(f"{num} is not a Prime Number")
```

q718.py

Ce programme Python définit une classe PrimeChecker qui peut être utilisée pour vérifier si un nombre donné est premier ou non. Voici une explication du fonctionnement du programme :

La classe PrimeChecker est définie et prend un entier num comme paramètre lors de la création d'une instance de la classe. La méthode __init__ est le constructeur de la classe et initialise la variable d'instance self.num avec la valeur qui lui a été transmise. La méthode is_prime est utilisée pour déterminer si le nombre stocké dans self.num est premier ou non. Elle suit les étapes suivantes : Si le nombre est inférieur ou égal à 1, elle renvoie False, car les nombres premiers sont définis comme des entiers positifs supérieurs à 1. Si le nombre est égal à 2, il renvoie True, car 2 est le seul nombre premier pair. Si le nombre est pair (c'est-à-dire divisible par 2), il renvoie False car les nombres premiers (autres que 2) sont toujours impairs. Il vérifie

ensuite la divisibilité du nombre à partir de 3 jusqu'à la racine carrée du nombre. Il effectue cette opération dans une boucle avec un pas de 2 pour ne vérifier que les nombres impairs, car les nombres pairs supérieurs à 2 ne peuvent pas être premiers. S'il trouve un diviseur dans cet intervalle, il renvoie False, indiquant que le nombre n'est pas premier. Si aucune des conditions ci-dessus n'est remplie, il renvoie True, ce qui indique que le nombre est premier. Le programme prend ensuite en compte la saisie d'un nombre par l'utilisateur à l'aide de input() et le convertit en un nombre entier à l'aide de int(). Ce nombre est stocké dans la variable num. Une instance de la classe PrimeChecker est créée avec les données de l'utilisateur et est affectée à la variable checker. La méthode is_prime de l'instance de checker est appelée. Si la méthode renvoie True, cela signifie que le nombre est premier et un message indiquant que le nombre est premier est imprimé. Si la méthode renvoie False, cela signifie que le nombre n'est pas premier et un message correspondant est imprimé.

Question 10

Écrire un programme python pour compter le nombre d'objets créés

Code python :

```python
class Student:
  # Class variable to keep track of the number of objects created
  count = 0

  def __init__(self,name,age):
    self.name = name
    self.age = age
    Student.count += 1 # Increment the count when an object is created

  def GetDetails(self):
    print("Name :",self.name)
    print("Age :",self.age)

# Create instances of MyClass
s1 = Student("Sam Kumar",21)
s2 = Student("Tiya",20)
s3 = Student("Sathish",19)
s3 = Student("Deepika",21)

# Print the number of objects created
print("Number of Objects : ", Student.count)
```

q719.py

Ce programme Python définit une classe appelée Student et démontre l'utilisation des variables de classe, de la création d'objets et des méthodes d'instance. Voici une explication du programme :

classe Étudiant : : Cette ligne définit une classe nommée Student. count = 0 : Il s'agit d'une variable de classe nommée count qui est utilisée pour garder une trace

du nombre d'objets Student créés. Elle est initialisée à 0. def __init__(self, name, age) : : Il s'agit de la méthode de construction de la classe Étudiant. Elle est utilisée pour initialiser les attributs d'un objet Étudiant. self est une référence à l'instance de la classe en cours de création, et name et age sont les attributs qui sont définis pour chaque objet. self.name = name et self.age = age : Ces lignes définissent les attributs nom et âge de l'objet Étudiant avec les valeurs passées en argument. Student.count += 1 : Cette ligne incrémente la variable de classe count de 1 à chaque fois qu'un nouvel objet Etudiant est créé. Cela permet de garder une trace du nombre d'objets Étudiant créés. def GetDetails(self) : : Il s'agit d'une méthode d'instance de la classe Étudiant, utilisée pour imprimer le nom et l'âge d'un étudiant. print("Name :", self.name) et print("Age :", self.age) : Ces lignes sont utilisées pour imprimer le nom et l'âge d'un objet Étudiant lorsque la méthode GetDetails est appelée. Quatre instances de la classe Étudiant sont créées : s1 = Student("Sam Kumar", 21) : un objet Student est créé avec le nom "Sam Kumar" et l'âge 21. s2 = Student("Tiya", 20) : un autre objet Student est créé avec le nom "Tiya" et l'âge de 20 ans. s3 = Student("Sathish", 19) : Un autre objet Student est créé avec le nom "Sathish" et l'âge 19. s3 = Étudiant("Deepika", 21) : Un autre objet Etudiant est créé avec le nom "Deepika" et l'âge de 21 ans. Notez que cette ligne réaffecte la variable s3, écrasant l'objet précédent créé avec cette variable. print("Nombre d'objets : ", Student.count) : Cette ligne imprime le nombre total d'objets Étudiant créés en accédant à la variable de classe count. Elle reflète le nombre d'objets créés, qui dans ce cas est de 3 puisque la variable s3 a été réaffectée.

Question 11

Ecrire un programme python pour vérifier le numéro Armstrong en utilisant une approche orientée objet.

Code python :

```python
class ArmstrongChecker:
    def __init__(self, num):
        self.num = num

    def is_armstrong(self):
        num_str = str(self.num)  # Convert the number to a string to count the digits
        num_digits = len(num_str)  # Calculate the number of digits
        digit_sum = sum(int(digit) ** num_digits for digit in num_str)  # Calculate the sum of the nth powers of its digits
        return digit_sum == self.num  # Check if the sum is equal to the original number

num = int(input("Enter a Number : "))

checker = ArmstrongChecker(num)

if checker.is_armstrong():
    print(f"{num} is an Armstrong Number")
else:
    print(f"{num} is not an Armstrong Number")
```

q720.py

Ce programme Python définit une classe appelée ArmstrongChecker qui vérifie si un nombre donné est un nombre Armstrong. Voici une explication pas à pas de son fonctionnement :

La classe ArmstrongChecker est définie par un constructeur __init__, qui prend un argument, num, représentant le nombre à vérifier. Dans le constructeur, self.num se voit attribuer la valeur de l'argument num, qui stocke le nombre à vérifier pour être un nombre Armstrong. La classe possède une méthode appelée is_armstrong(). Cette méthode vérifie si le nombre stocké dans self.num est un nombre d'Armstrong et renvoie une valeur booléenne (True ou False). A l'intérieur de la méthode is_armstrong : a. num_str est calculé en convertissant le nombre en une chaîne de caractères. Cela permet de compter le nombre de chiffres dans le nombre original. num_digits est calculé comme la longueur de num_str, ce qui donne le nombre de chiffres dans le nombre original. digit_sum est calculé à l'aide d'une expression de générateur. Il calcule la somme de chaque chiffre élevé à la puissance de num_chiffres. Par exemple, si num est 153, digit_sum sera calculé comme suit : $(1^3 + 5^3 + 3^3)$, soit 153. Enfin, la méthode renvoie True si digit_sum est égal au nombre original self.num, ce qui indique que le nombre est un nombre Armstrong. Dans le cas contraire, elle renvoie False. Le programme prend ensuite en compte la saisie d'un nombre par l'utilisateur en utilisant int(input("Enter a Number : ")) et le stocke dans la variable num. Une instance de la classe ArmstrongChecker est créée avec le nombre fourni num comme argument, et l'instance est stockée dans la variable checker. Le programme vérifie ensuite si le nombre est un nombre Armstrong en appelant la méthode is_armstrong() de l'objet checker. Si le nombre est un nombre d'Armstrong (c'est-à-dire que la méthode renvoie True), il imprime un message indiquant que le nombre est un nombre

d'Armstrong. Dans le cas contraire, un message indiquant que le nombre n'est pas un nombre d'Armstrong s'affiche.

Le programme vérifie si un nombre donné est un nombre d'Armstrong en utilisant la définition du nombre d'Armstrong (un nombre est un nombre d'Armstrong si la somme de ses chiffres élevée à la puissance du nombre de chiffres est égale au nombre original). Pour ce faire, il encapsule la logique dans une classe appelée Armstrong-Checker.

Question 12

Écrire un programme python pour l'héritage multi-niveaux

Code python :

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

    def display_info(self):
        print("Name : ",self.name)
        print("Age : ",self.age)

class Student(Person):  # Intermediate class inheriting from Person
    def __init__(self, name, age, student_id):
        super().__init__(name, age)
        self.student_id = student_id

    def display_student_info(self):
        super().display_info()
        print("Student ID : ",self.student_id)


class GraduateStudent(Student):   # Derived class inheriting from
    ↪ Student
    def __init__(self, name, age, student_id, research_topic):
        super().__init__(name, age, student_id)
        self.research_topic = research_topic

    def display_graduate_info(self):
        super().display_student_info()
        print("Research Topic : ",self.research_topic)

# Create an instance of GraduateStudent
graduate_student = GraduateStudent("Alice", 25, "STU001", "Machine
    ↪ Learning")

# Display information using the multilevel inheritance
graduate_student.display_graduate_info()
```

q721.py

Ce programme Python définit une hiérarchie de classes avec un héritage à plusieurs niveaux en Python. Voici un aperçu des classes et de leurs relations :

Personne (classe de base) : Cette classe possède un constructeur __init__ qui initialise les attributs nom et âge. Elle possède également une méthode display_info qui imprime le nom et l'âge. Étudiant (classe intermédiaire) : Cette classe hérite de Person et ajoute un nouvel attribut student_id. Elle possède son propre constructeur, qui prend en paramètre le nom, l'âge et l'identifiant de l'étudiant. Le constructeur de la classe Student appelle le constructeur de la classe Person en utilisant super() pour définir le nom et l'âge. Elle possède également une méthode display_student_info, qui appelle la méthode display_info de la classe Person et ajoute les informations relatives à l'identifiant de l'étudiant. GraduateStudent (classe dérivée) : Cette classe

hérite de la classe Student et ajoute un nouvel attribut research_topic. Elle possède son propre constructeur, qui prend en paramètre le nom, l'âge, l'identifiant de l'étudiant et le sujet de la recherche. Le constructeur de la classe GraduateStudent appelle le constructeur de la classe Student en utilisant super() pour définir le nom, l'âge et l'identifiant de l'étudiant. Il possède également une méthode display_graduate_info, qui appelle la méthode display_student_info de la classe Student et ajoute les informations research_topic.

Vous créez une instance de la classe GraduateStudent avec le nom "Alice", l'âge de 25 ans, le numéro d'étudiant "STU001" et le sujet de recherche "Machine Learning". Vous appelez ensuite la méthode display_graduate_info sur l'objet graduate_student, qui imprime toutes les informations de la classe de base Person, de la classe Student et de la classe GraduateStudent, y compris le sujet de recherche.

Question 13

Ecrire un programme python pour vérifier le nombre de palindromes en utilisant une approche orientée objet.

Code python :

```python
class PalindromeChecker:
    def __init__(self, num):
        self.num = num

    def is_palindrome(self):
        num_str = str(self.num)
        reversed_str = num_str[::-1]  # Reverse the string
        return num_str == reversed_str  # Check if the reversed string is equal to the original string


num = int(input("Enter a Number : "))

checker = PalindromeChecker(num)

if checker.is_palindrome():
    print(f"{num} is a Palindrome Number")
else:
    print(f"{num} is not a Palindrome Number")
```

q722.py

Le programme Python PalindromeChecker est conçu pour vérifier si un nombre donné est un palindrome. Voici une explication du fonctionnement du programme :

class PalindromeChecker : Il s'agit d'une classe qui définit un vérificateur de palindromes. Elle possède deux méthodes : __init__(self, num) : Il s'agit de la méthode du constructeur qui initialise l'objet avec le nombre num. is_palindrome(self) : Cette méthode vérifie si le nombre est un palindrome. Pour ce faire, elle convertit d'abord le nombre en une chaîne de caractères (num_str). Ensuite, elle inverse la chaîne en

utilisant le découpage (reversed_str = num_str[: :-1]) et vérifie si la chaîne inversée est égale à la chaîne originale. Si c'est le cas, il renvoie True, ce qui indique que le nombre est un palindrome ; sinon, il renvoie False. num = int(input("Enter a Number : ")) : Cette ligne de code invite l'utilisateur à saisir un nombre et convertit la saisie de l'utilisateur en un nombre entier, qui est stocké dans la variable num. checker = PalindromeChecker(num) : Cette ligne crée une instance de la classe PalindromeChecker avec le nombre saisi par l'utilisateur. La valeur num est transmise comme argument au constructeur, initialisant l'attribut num de l'objet checker. if checker.is_palindrome() : : Cette instruction conditionnelle vérifie si la méthode is_palindrome de l'objet checker renvoie True. Si c'est le cas, cela signifie que le nombre saisi est un palindrome et un message indiquant que le nombre est un palindrome s'affiche. Dans le cas contraire, un message indiquant que le nombre n'est pas un palindrome est imprimé. Le programme imprime alors "num est un nombre palindrome" ou "num n'est pas un nombre palindrome" en fonction du résultat de la méthode is_palindrome.

Voici comment fonctionne le programme :

L'utilisateur est invité à saisir un nombre. Le programme crée une instance de la classe PalindromeChecker avec le nombre saisi. Il vérifie si le nombre saisi est un palindrome à l'aide de la méthode is_palindrome. En fonction du résultat, il imprime un message indiquant si le nombre est un palindrome ou non.

Ce programme démontre le concept d'utilisation d'une classe pour encapsuler la fonctionnalité permettant de vérifier si un nombre est un palindrome, ce qui rend le code plus organisé et réutilisable.

Question 14

Écrire un programme pour le programme d'enregistrement de la taille des étudiants pour une école en Python

Code python :

```python
class SchoolHeightRecords:
    def __init__(self):
        self.student_records = {}

    def add_student(self, student_name, height):
        self.student_records[student_name] = height

    def remove_student(self, student_name):
        if student_name in self.student_records:
            del self.student_records[student_name]
            print(f"{student_name} Records Remove Success")
        else:
            print(f"{student_name} Not Found in Records")

    def find_student_height(self, student_name):
        if student_name in self.student_records:
            return self.student_records[student_name]
        else:
            return None

    def display_records(self):
        print("Student Height Records:")
        for student, height in self.student_records.items():
            print(f"{student} : {height} cm")


school_records = SchoolHeightRecords()

print("1. Add Student Height")
print("2. Remove Student Height")
print("3. Find Student Height")
print("4. Display All Records")
print("5. Quit")

while True:

    choice = int(input("\nEnter your Choice : "))

    if choice == 1:
        student_name = input("Enter Student Name : ")
        height = input("Enter Student Height (in cm) : ")
        school_records.add_student(student_name.upper(), height)
        print(f"{student_name}'s Height Added to Records.")
    elif choice == 2:
        student_name = input("Enter Student Name to Remove : ")
        school_records.remove_student(student_name.upper())
    elif choice == 3:
        student_name = input("Enter Student Name to find Height : ")
        height =
        ↪  school_records.find_student_height(student_name.upper())
        if height is not None:
            print(f"{student_name}'s Height : {height} cm")
        else:
            print(f"{student_name} Not Found in Records")
    elif choice == 4:
        school_records.display_records()
```

Question 15

Ecrire un programme python pour gérer l'enregistrement d'un magasin de téléphonie (mobile shop) en utilisant la classe

Code python :

```python
class Phone:
    def __init__(self, brand, model, price):
        self.brand = brand
        self.model = model
        self.price = price

class PhoneStore:
    def __init__(self):
        self.inventory = []

    def add_phone(self, phone):
        self.inventory.append(phone)

    def remove_phone(self, brand, model):
        for phone in self.inventory:
            if phone.brand == brand and phone.model == model:
                self.inventory.remove(phone)
                print(f"Removed {brand} {model} from inventory.")
                return
        print(f"{brand} {model} not found in inventory.")

    def find_phone(self, brand, model):
        for phone in self.inventory:
            if phone.brand == brand and phone.model == model:
                return phone
        return None

    def display_inventory(self):
        print("Phone Store Inventory :")
        for phone in self.inventory:
            print(f"Brand : {phone.brand}, Model : {phone.model}, Price
            ↪  : ${phone.price:.2f}")


phone_store = PhoneStore()

print("1. Add Phone to Inventory")
print("2. Remove Phone from Inventory")
print("3. Find Phone in Inventory")
print("4. Display Inventory")
print("5. Quit")

while True:
    choice = input("\nEnter your Choice : ")

    if choice == "1":
        brand = input("Enter Phone Brand : ")
        model = input("Enter Phone Model : ")
        price = float(input("Enter Phone Price : "))
        phone = Phone(brand.capitalize(), model.capitalize(), price)
        phone_store.add_phone(phone)
        print(f"Added {brand} {model} to inventory.")
    elif choice == "2":
        brand = input("Enter Phone Brand to Remove : ")
        model = input("Enter Phone Model to Remove : ")
        phone_store.remove_phone(brand.capitalize(),
```

Question 16

Écrire un programme python pour additionner deux distances en utilisant les concepts de classe et d'objet

Code python :

```python
class Distance:
    def __init__(self, km=0, m=0, cm=0):
        self.km = km
        self.m = m
        self.cm = cm

    def add(self, other_distance):
        total_km = self.km + other_distance.km
        total_m = self.m + other_distance.m
        total_cm = self.cm + other_distance.cm

        # Adjust units if necessary
        if total_cm >= 100:
            total_m += total_cm // 100
            total_cm %= 100
        if total_m >= 1000:
            total_km += total_m // 1000
            total_m %= 1000

        return Distance(total_km, total_m, total_cm)

    def display(self):
       return f"{self.km} KM {self.m} M {self.cm} CM"

print("Enter First Distance")
km1 = int(input("Enter Kilometers : "))
m1 = int(input("Enter Meters : "))
cm1 = int(input("Enter Centimeters : "))
distance1 = Distance(km1, m1, cm1)

print("\nEnter Second Distance")
km2 = int(input("Enter Kilometers : "))
m2 = int(input("Enter Meters : "))
cm2 = int(input("Enter Centimeters : "))
distance2 = Distance(km2, m2, cm2)

# Add the distances
result_distance = distance1.add(distance2)

# Display the result
print("Sum of both Distances is :",result_distance.display())
```

q725.py

Ce programme Python définit une classe Distance qui représente une distance en kilomètres, mètres et centimètres. La classe fournit des méthodes permettant d'additionner deux distances et d'afficher une distance dans un format lisible par l'homme. Voici une explication du programme :

classe Distance : Il s'agit de la classe principale représentant une distance. Elle possède les méthodes suivantes : __init__(self, km=0, m=0, cm=0) : Le constructeur initialise l'objet avec des valeurs optionnelles pour les kilomètres (0 par défaut), les mètres (0 par défaut) et les centimètres (0 par défaut). add(self, other_distance) : Cette méthode prend en paramètre un autre objet Distance, ajoute les composantes correspondantes (kilomètres, mètres et centimètres) et renvoie un nouvel objet Distance. display(self) : Cette méthode renvoie une chaîne de caractères formatée représentant la distance. Le programme invite ensuite l'utilisateur à saisir deux distances : une pour la distance1 et une pour la distance2. L'utilisateur est invité à saisir la distance en kilomètres, en mètres et en centimètres pour chacune d'entre elles. Les objets Distance distance1 et distance2 sont créés avec les valeurs fournies par l'utilisateur. La méthode add est appelée sur distance1, en passant distance2 comme paramètre. Il en résulte un nouvel objet Distance result_distance qui représente la somme des deux distances d'entrée. Le programme affiche le résultat en appelant la méthode display sur distance_résultat, qui fournit la somme des deux distances dans un format lisible par l'homme.

Lorsque vous exécutez ce programme, il vous permet de saisir deux distances, puis de calculer et d'afficher la somme de ces distances. Le programme se charge d'ajuster les unités si nécessaire, en veillant à ce que le résultat soit affiché correctement en termes de kilomètres, de mètres et de centimètres.

Question 17

Ecrire un programme python pour trouver la personne la plus âgée de deux personnes en utilisant la classe et l'objet.

Code python :

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

def find_elder(person1, person2):
    if person1.age > person2.age:
        return person1
    elif person2.age > person1.age:
        return person2
    else:
        return None

# Create two Person objects
person1 = Person("Sathish", 27)
person2 = Person("Pooja", 25)

# Find the elder person
elder = find_elder(person1, person2)

if elder is None:
    print("Both persons are of the same age")
else:
    print(f"Elder Person \nName : {elder.name} \nAge : {elder.age}")
```

q726.py

Ce programme Python définit une classe Person et une fonction find_elder pour trouver la personne la plus âgée parmi deux individus donnés en fonction de leur âge. Voici une explication du programme :

classe Personne : Il s'agit de la classe Personne avec un constructeur __init__ qui initialise deux attributs, le nom et l'âge. Elle représente une personne avec un nom et un âge. def find_elder(personne1, personne2) : Cette fonction prend deux objets Person, person1 et person2, comme paramètres. Elle compare leurs âges et renvoie la personne la plus âgée. Si les deux personnes ont le même âge, elle renvoie None. Le programme crée deux objets Personne, personne1 et personne2, représentant deux individus avec leurs noms et âges respectifs. La fonction find_elder est appelée avec ces deux objets Personne et le résultat est stocké dans la variable elder. Le programme vérifie ensuite si elder est None. Si c'est le cas, cela signifie que les deux personnes ont le même âge, et il imprime donc "Les deux personnes ont le même âge". Sinon, il imprime le nom et l'âge de la personne la plus âgée.

Lorsque vous exécutez ce programme, il détermine laquelle des deux personnes est la plus âgée en fonction de leur âge et imprime le message correspondant. Si les deux personnes ont le même âge, il indiquera que les deux ont le même âge. Ce programme montre comment définir une classe, créer des objets et comparer leurs attributs pour prendre des décisions basées sur l'état de l'objet.

Question 18

Écrire un programme python pour le système de gestion bancaire

Code python :

```python
class Bank:
    def __init__(self):
        self.accounts = {}

    def create_account(self, account_number, account_holder, initial_balance):
        if account_number in self.accounts:
            return "Account already exists"
        if initial_balance < 0:
            return "Initial balance must be non-negative"

        self.accounts[account_number] = {
            'account_holder': account_holder,
            'balance': initial_balance
        }
        return "Account created successfully"

    def deposit(self, account_number, amount):
        if account_number not in self.accounts:
            return "Account does not exist"
        if amount <= 0:
            return "Amount to deposit must be positive"

        self.accounts[account_number]['balance'] += amount
        return "Deposited " + str(amount) + " successfully. New balance : " + str(self.accounts[account_number]['balance'])

    def withdraw(self, account_number, amount):
        if account_number not in self.accounts:
            return "Account does not exist"
        if amount <= 0:
            return "Amount to withdraw must be positive"

        if self.accounts[account_number]['balance'] < amount:
            return "Insufficient balance."

        self.accounts[account_number]['balance'] -= amount
        return "Withdrew " + str(amount) + " successfully. New balance : " + str(self.accounts[account_number]['balance'])

    def check_balance(self, account_number):
        if account_number not in self.accounts:
            return "Account does not exist"

        return "Account Holder : " + self.accounts[account_number]['account_holder'] + "\nBalance : " + str(self.accounts[account_number]['balance'])

bank = Bank()  # Create a Bank object

print("\n****** Bank Management System ******")
print("\n1. Create Account")
print("2. Deposit Money")
print("3. Withdraw Money")
print("4. Check Balance")
```

Question 19

Ecrire un programme python pour obtenir les détails de l'étudiant en entrée et imprimer le résultat après la mise à jour des notes.

Code python :

```python
class Student:
  def __init__(self):
    self.student_details = {}

  @property
  def input_student_details(self):  # input student details
    roll_number = input("Enter Roll Number : ")
    name = input("Enter Student Name : ")
    marks = float(input("Enter Marks : "))
    self.student_details[roll_number] = {'name': name, 'marks': marks}
    print("Marks Added successfully")

  def update_student_marks(self):    # update student marks
    roll_number = input("Enter Roll Number of the Student to Update
    ↪  Marks : ")
    if roll_number in self.student_details:
      new_marks = float(input("Enter New Marks : "))
      self.student_details[roll_number]['marks'] = new_marks
      print("Marks updated successfully")
    else:
      print("Student not found")

  def print_student_details(self):  # print student details
    roll_number = input("Enter Roll Number of the student to view
    ↪  details : ")
    if roll_number in self.student_details:
      student = self.student_details[roll_number]
      print("Roll Number :", roll_number)
      print("Student Name :", student['name'])
      print("Marks :", student['marks'])
    else:
      print("Student not found")

obj = Student()

while True:
    print("\n ********* Student Management System *********  ")
    print("\n1. Input Student Details")
    print("2. Update Student Marks")
    print("3. Print Student Details")
    print("4. Exit")

    choice = input("\nEnter your Choice (1 to 4): ")

    if choice == '1':
        obj.input_student_details
    elif choice == '2':
        obj.update_student_marks()
    elif choice == '3':
        obj.print_student_details()
    elif choice == '4':
        print("Exiting the program")
        break
    else:
        print("Invalid Choice. Please Try Again !!!")
```

Question 20

Ecrire un programme python pour les tableaux d'objets

Code python :

```python
class Student:
    def __init__(self, roll_number, name, marks):
        self.roll_number = roll_number
        self.name = name
        self.marks = marks

    def __str__(self):
        return f"Roll Number : {self.roll_number}\nStudent Name : {self.name}\nMarks : {self.marks}"

students = [] # Create an array (list) of Student objects

students.append(Student(101, "Mithra", 85.5)) # Add students to the array
students.append(Student(102, "Akhil", 78.0))
students.append(Student(103, "Sathish", 92.5))

for student in students: # print individual student details
    print(student,"\n")

# Access and modify attributes of a student
student = students[0]
print("Before Update : ",student)
student.name = "Pooja"
student.marks = 90.0
print("\nAfter Update : ",student)
```

q729.py

Le programme Python définit une classe d'étudiants et crée une liste (étudiants) pour stocker les instances de la classe d'étudiants. Il montre également comment accéder aux attributs d'un étudiant et les modifier. Voici la décomposition du code :

classe Étudiant : Cette classe définit une classe appelée Student, qui représente un étudiant avec des attributs tels que le numéro de rôle, le nom et les notes. __init__(self, roll_number, name, marks) : La méthode du constructeur initialise un objet Student avec le numéro de rôle, le nom et les notes fournis. __str__(self) : Cette méthode spéciale renvoie une représentation sous forme de chaîne de caractères de l'objet Étudiant, affichant le numéro de rôle, le nom et les notes. étudiants = [] : Cette méthode crée une liste vide pour stocker les objets Étudiant. Trois étudiants sont créés et ajoutés à la liste des étudiants à l'aide de la méthode append. Chaque étudiant a un numéro de rôle, un nom et des notes uniques. Une boucle for itère sur la liste des étudiants et imprime les détails de chaque étudiant à l'aide de la méthode __str__. Elle montre ensuite comment accéder aux attributs d'un étudiant spécifique et les modifier. Dans ce cas, il sélectionne le premier étudiant de la liste, modifie son nom

et ses notes, et imprime les informations mises à jour.

Question 21

Ecrire un programme python pour rechercher des objets dans un tableau d'objets en utilisant l'ID

Code python :

```python
class Student:
    def __init__(self, student_id, name, per):
        self.student_id = student_id
        self.name = name
        self.per = per

    def __str__(self):
        return "Student ID : " + str(self.student_id) + "\nStudent Name
        ↪  : " + self.name + "\nPercentage : " + str(self.per)

# Create an array (list) of Student objects
students = [
      Student(101, "Mithra", 85.5),
      Student(102, "Akhil", 78.0),
      Student(103, "Tiya", 92.5),
      Student(104, "Ramesh", 84.5),
      Student(105, "Sam Kumar", 65.5)
     ]

def search_student_by_id(student_id):  # search for a student by ID
    for student in students:
        if student.student_id == student_id:
            return student
    return None

# Print all student IDs
print("*** Student IDs ***")
for student in students:
    print(student.student_id)

search_id = int(input("\nEnter Student ID to Search : "))

found_student = search_student_by_id(search_id) # Search for the
↪  student by ID

if found_student:
    print("Student Found")
    print(found_student)
else:
    print("Student Not Found")
```

q730.py

Le programme définit une classe Étudiant, crée une liste d'objets Étudiant et vous permet de rechercher un étudiant par son numéro d'identification. Voici la décomposition du code :
classe Étudiant : Cette classe représente un étudiant avec des attributs tels que l'identifiant de l'étudiant, le nom et le pourcentage. __init__(self, student_id, name, per) : Le constructeur initialise un objet Student avec l'identifiant, le nom et le pourcentage fournis. __str__(self) : Cette méthode spéciale renvoie une représentation

sous forme de chaîne de caractères de l'objet Étudiant, affichant l'identifiant, le nom et le pourcentage de l'étudiant. Un tableau (liste) d'objets Étudiant est créé et initialisé avec les données de l'étudiant. search_student_by_id(student_id) : Cette fonction permet de rechercher un étudiant à partir de son numéro d'identification. Elle parcourt la liste des étudiants et renvoie le premier objet étudiant correspondant à l'identifiant spécifié, ou Aucun si aucune correspondance n'est trouvée. Le code imprime tous les identifiants des étudiants de la liste à l'aide d'une boucle for. L'utilisateur est invité à saisir l'identifiant de l'étudiant à rechercher. La fonction search_student_by_id est appelée avec l'identifiant fourni par l'utilisateur et, si un étudiant correspondant à cet identifiant est trouvé, il est imprimé avec un message "Étudiant trouvé". Si aucun étudiant correspondant n'est trouvé, un message "Étudiant non trouvé" s'affiche.

Question 22

Écrire un programme python pour l'initialisation du constructeur

Code python :

```python
class Student:
    def __init__(self, roll_number, name, percent):
        self.roll_number = roll_number
        self.name = name
        self.percent = percent

    def display_details(self):
        print("Roll Number :", self.roll_number)
        print("Name :", self.name)
        print("Percentage :", self.percent)

# Creating objects of the Student class with constructor initialization
s1 = Student(501, "Kim", 85.5)
s2 = Student(502, "Bob", 78.0)
s3 = Student(503, "Tin", 90.3)

print("Student 1 Details :")
s1.display_details()

print("\nStudent 2 Details :")
s2.display_details()

print("\nStudent 3 Details :")
s3.display_details()
```

q731.py

Le programme définit une classe Étudiant et crée trois instances de la classe, chacune représentant un étudiant différent. Il affiche ensuite les détails de chaque étudiant à l'aide de la méthode display_details. Voici la décomposition du code :

classe Étudiant : Cette classe définit un étudiant avec des attributs tels que le numéro de rôle, le nom et le pourcentage. __init__(self, roll_number, name, percent) : Le constructeur initialise un objet Student avec le numéro de rôle, le nom et le pourcentage fournis. display_details(self) : Cette méthode permet d'afficher les détails d'un étudiant, y compris son numéro de rôle, son nom et son pourcentage. Trois instances de la classe Étudiant (s1, s2 et s3) sont créées et initialisées avec des données différentes pour chaque étudiant. Le code affiche ensuite les détails de chaque étudiant en appelant la méthode display_details pour chaque instance.

Voici ce que fait le code :

Il crée trois objets étudiants avec des données différentes. Il imprime les détails de chaque élève à l'aide de la méthode display_details, en affichant leur numéro de rôle, leur nom et leur pourcentage.

Question 23

Ecrire un programme python pour paramétrer un constructeur et un destructeur.

Code python :

```python
class Student:
  def __init__(self, roll_number, name, per):
    self.roll_number = roll_number
    self.name = name
    self.per = per

  def display_details(self):
    print("Roll Number :", self.roll_number)
    print("Name :", self.name)
    print("Percentage :",self.per)

  def __del__(self):
    print("Deleting student with Roll Number", self.roll_number)

# Creating objects of the Student class with a parameterized
↪  constructor
s1 = Student(501, "Kim", 85.5)
s2 = Student(502, "Bob", 78.0)
s3 = Student(503, "Tin", 90.3)

print("Student 1 Details :")
s1.display_details()

print("\nStudent 2 Details :")
s2.display_details()

print("\nStudent 3 Details :")
s3.display_details()

# Deleting student objects (calling the destructor)
del s1
del s2
del s3
```

q732.py

Le programme définit une classe Étudiant avec un constructeur et un destructeur. Il crée trois instances de la classe, affiche leurs détails, puis supprime explicitement ces instances pour démontrer la fonctionnalité du destructeur. Voici la décomposition du code :

classe Étudiant : Cette classe définit un étudiant avec des attributs tels que le numéro de rôle, le nom et le pourcentage. __init__(self, roll_number, name, per) : Le constructeur initialise un objet Student avec le numéro de rôle, le nom et le pourcentage fournis. display_details(self) : Cette méthode est utilisée pour imprimer les détails d'un étudiant, y compris son numéro de rôle, son nom et son pourcentage. del__(self) : Cette méthode spéciale est le destructeur. Elle est automatiquement appelée lorsqu'une instance de la classe est supprimée. Dans ce cas, elle imprime un message indiquant le numéro de rôle de l'étudiant supprimé. Trois instances de la classe Étudiant (s1, s2 et s3) sont créées et initialisées avec des données différentes

pour chaque étudiant. Le code affiche les détails de chaque étudiant en appelant la méthode display_details pour chaque instance. Les instances sont explicitement supprimées à l'aide de l'instruction del, qui déclenche la méthode du destructeur __del__.

Voici ce que fait le code :

Il crée trois objets étudiants avec des détails différents. Il imprime les détails de chaque étudiant à l'aide de la méthode display_details. Il supprime explicitement chaque objet élève, ce qui déclenche le destructeur et imprime un message indiquant que l'élève est supprimé.

Question 24

Ecrire un programme python pour ajouter des objets '+' en utilisant l'opérateur

Code python :

```python
class ComplexNumber:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __str__(self):
        return str(self.real) + " + " + str(self.imag) + "i"

    def __add__(self, other):
        real_part = self.real + other.real  # Add the real and
        ↪  imaginary parts separately
        imag_part = self.imag + other.imag
        return ComplexNumber(real_part, imag_part)

# Create two complex number objects
complex1 = ComplexNumber(3, 2)
complex2 = ComplexNumber(5, 3)

# Add the complex numbers using the + operator
result = complex1 + complex2

print("Addition :",result)
```

q733.py

Le programme python définit une classe ComplexNumber qui représente les nombres complexes. Elle permet de créer des objets de type nombre complexe, de les afficher sous forme de chaînes de caractères et d'effectuer l'addition de deux nombres complexes à l'aide de l'opérateur +. Voici la décomposition du code :

classe ComplexNumber : Cette classe représente un nombre complexe avec les attributs real et imag, qui représentent respectivement les parties réelle et imaginaire. __init__(self, real, imag) : Le constructeur initialise un objet ComplexNumber avec les parties réelle et imaginaire fournies. __str__(self) : Cette méthode spéciale ren-

voie une représentation sous forme de chaîne de caractères du nombre complexe au format "a + bi" où "a" est la partie réelle, "b" la partie imaginaire et "i" l'unité imaginaire. __add__(self, other) : Cette méthode spéciale surcharge l'opérateur +, ce qui permet d'additionner deux nombres complexes. Elle additionne les parties réelle et imaginaire séparément et renvoie un nouvel objet ComplexNumber représentant la somme. Deux objets de nombres complexes, complex1 et complex2, sont créés avec des parties réelles et imaginaires différentes. Le code utilise l'opérateur + pour additionner complex1 et complex2, ce qui produit un nouvel objet ComplexNumber appelé result. L'instruction print affiche le résultat de l'addition, qui est la somme des nombres complexes.

Voici ce que fait le code :

Il crée deux objets nombres complexes avec des parties réelles et imaginaires différentes. Il additionne ces nombres complexes à l'aide de l'opérateur +, qui appelle la méthode __add__. Il imprime le résultat de l'addition.

Question 25

Ecrire un programme python pour comparer deux objets en utilisant l'opérateur '>'.

Code python :

```python
class Number:
    def __init__(self, value):
        self.value = value

    def __str__(self):
        return str(self.value)

    def __gt__(self, other):
        return self.value > other.value

# Create two Number objects
number1 = Number(5)
number2 = Number(3)

# Compare the two numbers using the > operator in an expression
res = number1 > number2

if res:
    print(number1, "is greater than", number2)
else:
    print(number1, "is not greater than", number2)
```

q734.py

Le programme python définit une classe Number qui représente un nombre. Il permet de créer des objets Number, de les afficher sous forme de chaînes et d'effectuer une comparaison supérieure à l'aide de l'opérateur >. Voici la décomposition du code :

classe Number : Cette classe représente un nombre avec une valeur d'attribut. __init__(self,

value) : Le constructeur initialise un objet Number avec la valeur fournie. __str__(self) : Cette méthode spéciale renvoie une représentation sous forme de chaîne de caractères du nombre. __gt__(self, other) : Cette méthode spéciale surcharge l'opérateur > (plus grand que), ce qui permet de comparer deux objets Number. Elle vérifie si la valeur de l'objet courant est supérieure à la valeur de l'autre objet. Deux objets Number, number1 et number2, sont créés avec des valeurs différentes. Le code compare number1 et number2 en utilisant l'opérateur > dans une expression et stocke le résultat dans la variable res. En fonction du résultat, le code imprime un message indiquant si nombre1 est supérieur à nombre2.

Voici ce que fait le code :

Il crée deux objets Number avec des valeurs différentes. Il compare les deux nombres à l'aide de l'opérateur >. Il imprime un message basé sur le résultat de la comparaison.

Question 26

Écrire un programme python pour passer des objets en tant qu'arguments et renvoyer des objets à partir d'une fonction

Code python :

```python
class MyClass:
  def __init__(self, value):
    self.value = value

def process_object(obj):
  obj.value *= 2  # Modify the object or perform some operations
  return obj   # Return the modified object

my_obj = MyClass(10)  # Create an object of MyClass

result_obj = process_object(my_obj) # Call the function and pass the
↪  object as an argument

print("Original Object :",my_obj.value)
print("Modified Object :",result_obj.value)
```

q735.py

Le programme python définit une classe MyClass et une fonction process_object. La fonction prend en argument un objet de la classe MyClass, modifie l'objet en doublant sa valeur et renvoie l'objet modifié. Voici la décomposition du code :

classe MaClasse : Cette classe représente un objet avec une valeur d'attribut. __init__(self, value) : Le constructeur initialise un objet MyClass avec la valeur fournie. def process_object(obj) : Cette fonction prend un objet obj comme argument, double l'attribut value de l'objet, puis renvoie l'objet modifié. Une instance de MyClass est créée et my_obj est initialisé avec une valeur de 10. La fonction process_object est appelée avec mon_obj comme argument. Elle modifie mon_obj en doublant son attribut value, et l'objet modifié est renvoyé et assigné à result_obj. Le code imprime l'attribut

de valeur de l'objet original (mon_obj) et de l'objet modifié (résultat_obj).

Voici ce que fait le code :

Il crée un objet MyClass avec une valeur initiale de 10. Il appelle la fonction process_object, qui modifie l'objet en doublant sa valeur. Il imprime l'attribut value de l'objet original et de l'objet modifié.

Question 27

Ecrire un programme python pour illustrer le fonctionnement des décorateurs

Code python :

```python
# Decorator function
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called")
        func()
        print("Something is happening after the function is called")
    return wrapper

# Function to be decorated
@my_decorator
def msg():
    print("Hello world !")

# Call the decorated function
msg()
```

q736.py

L'utilisation d'un décorateur en Python. Un décorateur est une fonction qui enveloppe une autre fonction pour lui ajouter des fonctionnalités. Dans ce code, vous avez défini une fonction décorateur my_decorator, qui enveloppe la fonction msg. Voici la décomposition du code :

def my_decorator(func) : Il s'agit de la fonction décoratrice. Elle prend une fonction func en argument et définit une fonction interne appelée wrapper. La fonction wrapper ajoute un comportement avant et après l'appel de func. def wrapper() : Il s'agit de la fonction interne du décorateur. Elle ajoute un comportement avant et après l'appel de la fonction originale func. À l'intérieur de la fonction wrapper, des instructions d'impression indiquent que quelque chose se passe avant et après l'appel de la fonction d'origine. return wrapper : La fonction decorator renvoie la fonction wrapper. @my_decorator : Il s'agit d'une syntaxe de décorateur, indiquant que la fonction msg est décorée avec mon_décorateur. Cela signifie que lorsque vous appelez msg, elle sera enveloppée par la fonction wrapper définie dans mon_décorateur. def msg() : Il s'agit de la fonction à décorer. Elle imprime simplement "Hello world !". Lorsque vous appelez msg(), vous appelez en fait la fonction décorée. Le décorateur my_decorator enveloppe msg, de sorte qu'il imprime les messages supplémentaires avant et après l'impression de "Hello world !

La fonction msg est décorée par le décorateur my_decorator, de sorte que lorsque vous appelez msg(), elle affiche "Quelque chose se passe avant l'appel de la fonction", puis "Hello world!", et enfin "Quelque chose se passe après l'appel de la fonction". Cela montre comment les décorateurs peuvent ajouter un comportement aux fonctions sans modifier leur code source.

Question 28

Ecrire un programme python pour illustrer le fonctionnement de la méthode abstraite

Code python :

```python
from abc import ABC, abstractmethod

class Shape(ABC):#Abstract base class
    @abstractmethod
    def area(self):
        pass

class Circle(Shape):# Create a subclass of Shape
    def __init__(self):
        self.radius = float(input("Enter the Radius :"))

    def area(self):
        return 3.1415 * self.radius * self.radius

class Rectangle(Shape):# Create a subclass of Shape
    def __init__(self):
        self.length = float(input("Enter the Length :"))
        self.width = float(input("Enter the Width :"))

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


# Create instances of the subclasses
cir = Circle()
rect = Rectangle()

# Call the area method on the objects
print("Area of Circle :", cir.area())
print("Area of Rectangle :", rect.area())
```

q737.py

L'utilisation des classes de base abstraites et de l'héritage en Python. Voici la décomposition du code :

from abc import ABC, abstractmethod : Cette ligne importe la classe ABC (Abstract Base Class) et le décorateur abstractmethod du module abc. La classe ABC est utilisée pour créer des classes de base abstraites, et le décorateur abstractmethod est

utilisé pour déclarer des méthodes abstraites dans ces classes. classe Shape(ABC) : Cette classe définit une classe de base abstraite Shape qui hérite de la classe ABC. Elle possède une zone de méthodes abstraites. @abstractmethod : Ce décorateur est utilisé pour déclarer la méthode area comme une méthode abstraite. Les méthodes abstraites doivent être implémentées par toute sous-classe concrète de la classe Shape. classe Circle(Forme) : Cette classe définit une sous-classe Circle de la classe Shape. Elle implémente la méthode area pour calculer la surface d'un cercle en fonction du rayon. classe Rectangle(Forme) : Cette classe définit une autre sous-classe Rectangle de la classe Shape. Elle met en œuvre la méthode area pour calculer la surface d'un rectangle en fonction de sa longueur et de sa largeur. Les instances des classes Cercle et Rectangle sont créées en invitant l'utilisateur à saisir les dimensions requises. La méthode area est appelée sur les objets Circle et Rectangle pour calculer et imprimer les surfaces du cercle et du rectangle, respectivement.

Voici ce que fait le code :

Il définit une classe de base abstraite Shape avec une méthode area abstraite. Il crée deux sous-classes concrètes, Circle et Rectangle, qui héritent de Shape et implémentent la méthode area. Elle invite l'utilisateur à saisir les dimensions d'un cercle et d'un rectangle. Il calcule et imprime les aires du cercle et du rectangle à l'aide des méthodes d'aire respectives.

Question 29

Ecrire un programme python pour convertir les heures en jours

Code python :

```python
class HoursToDaysConverter:
    def __init__(self, hours):
        self.hours = hours

    def convert_to_days(self):
        days = self.hours / 24
        return days
try:
    hours_input = float(input("Enter the Number of Hours : "))# Get the
    ↪  number of hours from the user
except ValueError:
    print("Invalid input. Please enter a valid number of hours.")
else:
    obj = HoursToDaysConverter(hours_input) # Create an instance of the
    ↪  HoursToDaysConverter class
    days_result = obj.convert_to_days() #Call the Convert hours to days
    ↪  method
    print(hours_input, "hours is equal to", days_result, "days.") #
    ↪  Display the result
```

q738.py

Le programme python définit une classe HoursToDaysConverter, qui est utilisée pour

convertir un nombre d'heures en jours. Voici la décomposition du code :

class HoursToDaysConverter : Cette classe représente un convertisseur permettant de convertir des heures en jours. Elle possède une méthode __init__ pour initialiser l'attribut hours et une méthode convert_to_days pour effectuer la conversion. Dans le bloc try, le code tente d'obtenir le nombre d'heures de l'utilisateur à l'aide de la fonction input. Il convertit l'entrée de l'utilisateur en un nombre flottant et l'affecte à la variable hours_input. Si l'utilisateur saisit une valeur non valide (qui n'est pas un nombre), une exception ValueError est levée et un message d'erreur est imprimé. Si l'utilisateur saisit un nombre d'heures valide, une instance de la classe HoursToDaysConverter est créée à partir de la variable hours_input et la méthode convert_to_days est appelée pour effectuer la conversion. Le résultat, qui est le nombre de jours, est stocké dans la variable days_result. Enfin, le code imprime le nombre original d'heures saisi par l'utilisateur et le nombre équivalent de jours basé sur la conversion.

Voici ce que fait le code :

Il prend le nombre d'heures saisi par l'utilisateur et tente de le convertir en un nombre flottant. Si l'entrée est valide, il convertit le nombre d'heures en jours à l'aide de la classe HoursToDaysConverter. Il affiche ensuite le nombre d'heures original et le nombre de jours équivalent.

Question 30

Écrire un programme python pour rechercher des objets dans un tableau d'objets à l'aide de la méthode filter().

Code python :

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

# Create an array of Person objects
people = [
    Person("Kim", 21),
    Person("Sam", 30),
    Person("Charlie", 29),
    Person("Bob", 25),
    Person("Tiya", 27),
]

# Define a function to filter people older than a certain age
def is_older_than_25(person):
    return person.age > 25

# Use the filter() method to search for people older than 25
filtered_people = filter(is_older_than_25, people)

# Convert the filtered result to a list (optional)
filtered_people_list = list(filtered_people)

# Display the filtered results without f-strings
print("People older than 25 :")
for person in filtered_people_list:
    print("Name :", person.name)
    print("Age :", person.age)
```

q739.py

Le programme python définit une classe Personne et une liste d'objets Personne. Il utilise ensuite la fonction filter() pour filtrer les personnes de la liste qui ont plus de 25 ans. Voici la décomposition du code :

classe Personne : Cette classe définit une personne avec des attributs de nom et d'âge. Le code crée une liste d'objets Personne, les personnes, avec des noms et des âges différents. def is_older_than_25(person) : Cette fonction définit un critère de filtrage. Elle renvoie True si l'âge de la personne est supérieur à 25 ; sinon, elle renvoie False. filter(is_older_than_25, people) : La fonction filter() prend la fonction is_older_than_25 et la liste des personnes, et renvoie un itérable contenant uniquement les objets Person pour lesquels la fonction is_older_than_25 renvoie True. liste_personnes_filtrées = liste(personnes_filtrées) : Cette ligne convertit l'itérable filtré en liste. Bien que cette conversion soit facultative, elle est effectuée ici pour faciliter la lecture en boucle des résultats filtrés à plusieurs reprises. Le code parcourt ensuite la liste filtrée et imprime le nom et l'âge des personnes âgées de plus de 25 ans.

Voici ce que fait le code :

Il définit une classe Personne pour représenter les personnes avec leur nom et leur

âge. Il crée une liste d'objets Personne. Il utilise la fonction filter() pour filtrer et extraire les personnes âgées de plus de 25 ans. Il convertit le résultat du filtrage en une liste (facultatif). Il affiche les noms et les âges des personnes âgées de plus de 25 ans.