La programmation orientée objet.

I. Les paradigmes de programmation.

Le paradigme de programmation est la façon (parmi d'autres) d'approcher la programmation informatique et de formuler les solutions aux problèmes et leur formalisation dans un langage de programmation approprié.

a. Le paradigme impératif.

En informatique, la programmation impérative est un paradigme de programmation qui décrit les opérations en séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme.
Le programmeur décrit l'ordre dans lequel sont exécutées les instructions. Il a aussi la possibilité d'accéder directement à la mémoire contenant les données.
C'est le paradigme le plus courant et historiquement le plus ancien puisqu'il a été inventé conjointement à la création des premiers ordinateurs.

```
ma_liste = [1,4,56,7,8,9,12]
somme = 0
for i in range(len(ma_liste)):
    somme = somme + ma_liste[i]
print("la somme des élèments de ma liste est", somme)
```

Exemples de langages de programmation : C, Pascal, Cobol, Fortran

b. Le paradigme fonctionnel (ou aussi appelé procédural).

Ce paradigme prend son origine dans le langage mathématiques traitant des fonctions.
Une fonction accepte des données et produit des données.
Une fonction peut être soit « primitive », soit formée par une composition de fonctions. Il n'y a pas de séparation entre données et programmes.
Ce paradigme ne contient pas la notion de variable : un programme écrit dans un langage fonctionnel ne produit pas d'effets de bords. Nous étudierons de manière un peu plus approfondi ce paradigme plus tard dans l'année.

```
def somme(tableau):
    somme = 0
    for i in range(len(tableau)):
        somme = somme + ma_liste[i]
    return somme

def moyenne(tableau):
    return somme(tableau)/len(tableau)
```

Exemples de langage de programmation : Python, Ocaml, Haskell, ...

c. Le paradigme orienté objet.

La programmation orientée objet (POO), ou programmation par objet, est un paradigme de programmation informatique. Elle consiste en la définition et l'interaction de briques logicielles appelées objet.

Un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. Il possède une structure interne et un comportement, et il sait interagir avec ses pairs.

Il s'agit donc de représenter ces objets et leurs relations ; l'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes.
Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle.

Exemples de langage de programmation : Python, C++, Java, ...

Pour conclure :En fonction du problème posé, il est utile de réfléchir au paradigme de programmation qui sera le plus efficace pour y répondre et ainsi de choisir le langage adéquat.

II. La programmation orientée objet.

a. Histoire.

La programmation orientée objet a fait ses débuts dans les années 1960 avec les réalisations dans le langage Lisp. Cependant, elle a été formellement définies avec les langages Simula dans les années 70, puis avec SmallTalk.
À partir des années 1980, commence l'effervescence des langages à objets : C++ (1983), Objective-C (1984), Eiffel (1986), Common Lisp Object System (1988), etc. Les années 1990 voient l'âge d'or de l'extension de la programmation par objets dans les différents secteurs du développement logiciel.

b. Conception orientée objet.

En 1995,l'ingénieur américain Grady Booch propose 5 étapes dans l'établissement d'une conception orientée objet.

1. Identifier les objets et leurs attributs : on chercher à identifier les objets du monde « réel » que l'on voudra réaliser.
2. Identifier les opérations : on cherche ensuite à identifier les actions que l'objet subit de la part de son environnement et qu'il provoque sur son environnement.
3. Établir la visibilité : l'objet étant maintenant identifié par ses caractéristiques et ses opérations, on définira ses relations avec les autres objets
4. Établir l'interface : dés que la visibilité est acquise, on définit l'interface précise de l'objet avec le monde extérieur.
5. Implémenter les objets : la dernière étape consiste à implanter les objets en écrivant le code.

c. Les Classes.

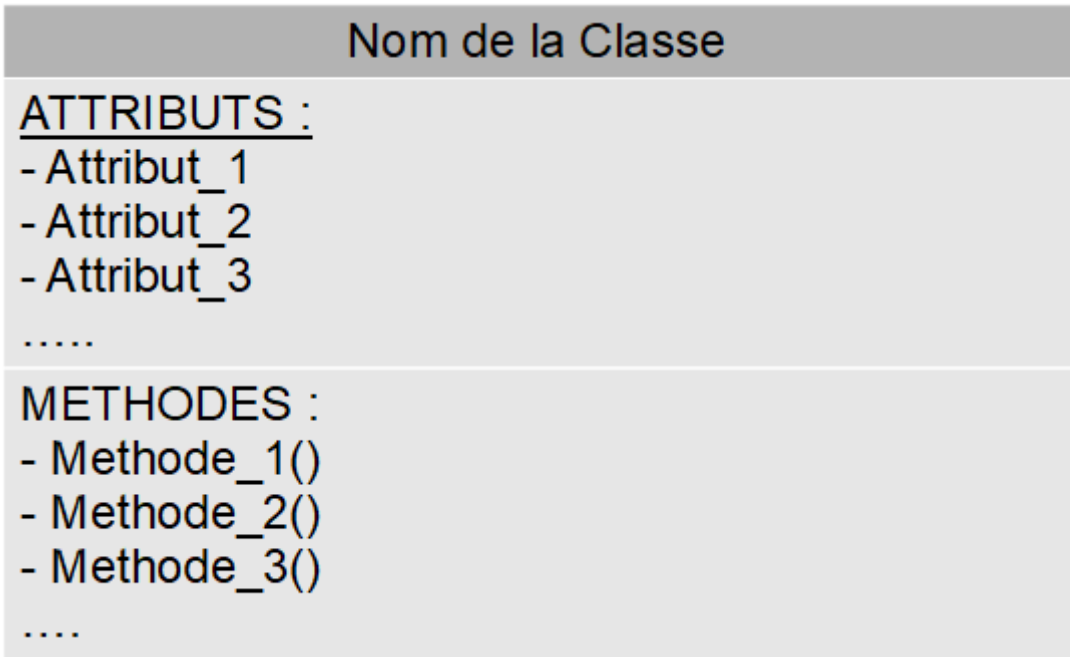
Dans la programmation orientée objet, les différents objets peuvent être construits indépendamment les uns des autres sans qu'il n'y ait de risque d'interférence.

Ce résultat est obtenu grâce au concept d'encapsulation : la fonctionnalité interne de l'objet et les variables qu'il utilise pour effectuer son travail, sont en quelque sorte, enfermées dans l'objet. Les autres objets et le monde extérieur ne peuvent y accéder qu'à travers des procédures bien définies, ce que l'on appelle l'interface de l'objet.

Un système complexe a un grand nombre d'objets. Pour réduire cette complexité, on regroupe les objets en classe.

Une classe est une description d'un ensemble d'objets ayant une structure de données commune (attributs) et pouvant réaliser des actions (méthodes). On considère en fait une classe comme un nouveau type de données. On appelle instance de la classe l'objet (du type de la classe) qui la représente.

On peut représenter graphiquement une classe de la manière suivante :



III. Les Classes en Python.

Python est conçu pour pouvoir travailler en programmation orientée objet.
Afin de créer une nouvelle classe d'objets Python, on utilise l'instruction class.

Dans cette partie, nous travaillerons sur un exemple : imaginons que nous voulions créer une classe d'objets CompteBancaire.
Ces objets auront trois attributs :

- nom : le nom du propriétaire du compte
- numéro : le numéro du compte en banque
- solde : la quantité d'argent présent sur le compte

Ces objets auront trois méthodes :

- depot (somme) : permettra d'ajouter une certaine somme au solde ;
- retrait(somme) : permettra de retirer une certaine somme au solde
- affiche() : permettra d'afficher le nom du titulaire du compte et le solde de son compte.

a. Conception de l'objet.

```
class CompteBancaire():
    def _init_(self,lenom,lenumero,lesolde=0):
        self.nom = lenom
        self.numero = lenumero
        self.solde = lesolde
```

Remarques:

- Le nom d'une classe commence toujours par une majuscule.

- les fonctions définies à l'intérieur d'une classe sont appelées méthodes ;
- une méthode possède nécessairement un paramètre appelé self en premier. C'est une référence d'instance, nous verrons cela plus bas. Ce paramètre n'apparaît pas lors de l'appel de la méthode ;
- une classe possède des attributs ou variables d'instance qui doivent être appelé avec le préfixe self. La classe CompteBancaire ci-dessus possède donc trois attributs nom, numero et solde. Dans notre exemple, le solde a été initialisé par défaut à 0.
- une classe possède une méthode spéciale **init**() nommée constructeur qui est appelée lorsqu'on créer une instance de cette classe.

b. Instanciation de l'objet.

```
>>> moncompte=CompteBancaire("Guichard",101,800)
>>> moncompte
<__main__.CompteBancaire object at 0x036B9810>
>>> moncompte.nom
'Guichard'
>>> moncompte.numero
101
>>> moncompte.solde
800
```

moncompte est alors une instance de la classe CompteBancaire : c'est à dire un objet créé à partir de la classe CompteBancaire()
Les arguments « Guichard », 101 et 800 ont été envoyés au constructeur qui les a affecté aux bonnes variables d'instances.
self est un argument « fantôme ». On ne le passe pas en paramètre.
Il est possible d'accéder aux paramètres du compte en utilisant `moncompte.nom` , `moncompte.numero`, `moncompte.solde`.

Toutefois, il est préférable de ne pas procéder ainsi.

c. Les méthodes.

```
def get_nom(self):
        '''permet d'accéder à l'attribut nom de l'objet compte bancaire'''
        return self.nom

    def get_numero(self):
         '''permet d'accéder à l'attribut numero de l'objet compte bancaire'''
         return self.numero

    def get_solde(self):
```

```
        '''permet d'accéder à l'attribut solde de l'objet compte bancaire'''
        return self.solde

    def depot(self,somme):
        '''permet de modifier le solde du compte bancaire en ajoutant de l'argent
sur le compte'''
        self.solde += somme

    def retrait(self, somme):
        '''permet de modifier le solde du compte bancaire en retirant de l'argent
sur le compte'''
        self.solde -= somme

    def affiche(self):
        ''' permet d'afficher les informations concernant le compte'''

        print("Le compte détenu par M. ou Mme. "+self.nom+" a un solde de
"+str(self.solde)+"€")
```

- Il est d'usage de créer des méthodes permettant d'accéder aux attributs.
- Pour faire appel aux méthodes, on écrira :
  `instance.methode(argument)`

```
>>> moncompte=CompteBancaire("Guichard",101,800)
>>> moncompte.depot(500)
>>> moncompte.affiche()
Le compte détenu par M. ou Mme. Guichard a un solde de 1300€
```