

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | |
|  | **DOSSIER DE CONCEPTION** | | | | |  | |
|  |  | | | | | |  |
|  | | TP n°2 : Sujet n°4  Implémentation de concepts orientés objet en Python & Analyse de données | | |  | | |
|  | | **Nathan AMSELLEM**  **Valentin THEDON** | | |  | | |
|  | | 03/11/2022  —  Cours de Programmation Orientée Objet Avancée  —  Professeur : Hamid MCheick | | |  | | |
|  | | |  |  | | | |

Table des matières

[Question n°4 3](#_Toc118478577)

[Principes de conceptions 3](#_Toc118478578)

[Cas de figure 3](#_Toc118478579)

[Modélisation Objet 4](#_Toc118478580)

[Analyse de l’implémentation 5](#_Toc118478581)

[Polymorphisme 5](#_Toc118478582)

[Overriding 5](#_Toc118478583)

[Overloading 5](#_Toc118478584)

[Généricité 5](#_Toc118478585)

[Modularité 6](#_Toc118478586)

[Analyse de données 6](#_Toc118478587)

[Cas de figure 6](#_Toc118478588)

[Entrées 6](#_Toc118478589)

[Sorties 6](#_Toc118478590)

[Objectif de l’analyse de données 6](#_Toc118478591)

[Analyse de l’implémentation 6](#_Toc118478592)

[Pandas & Seaborn 6](#_Toc118478593)

[Numpy & Listes 6](#_Toc118478594)

[Conclusion 6](#_Toc118478595)

# Question n°4

## 

## Principes de conceptions

Dans ce TP il nous est demandé d’introduire des concepts de programmation objets en python et de montrer une étude de cas de ces concepts :

* Polymorphisme
* Surcharge des méthodes (overloading),
* Redéfinition des méthodes (overriding)
* Généricité
* Modularité

Nous tenterons donc à travers une étude de cas que nous avons imaginé de présenter la manière d’introduire ces concepts de programmation.

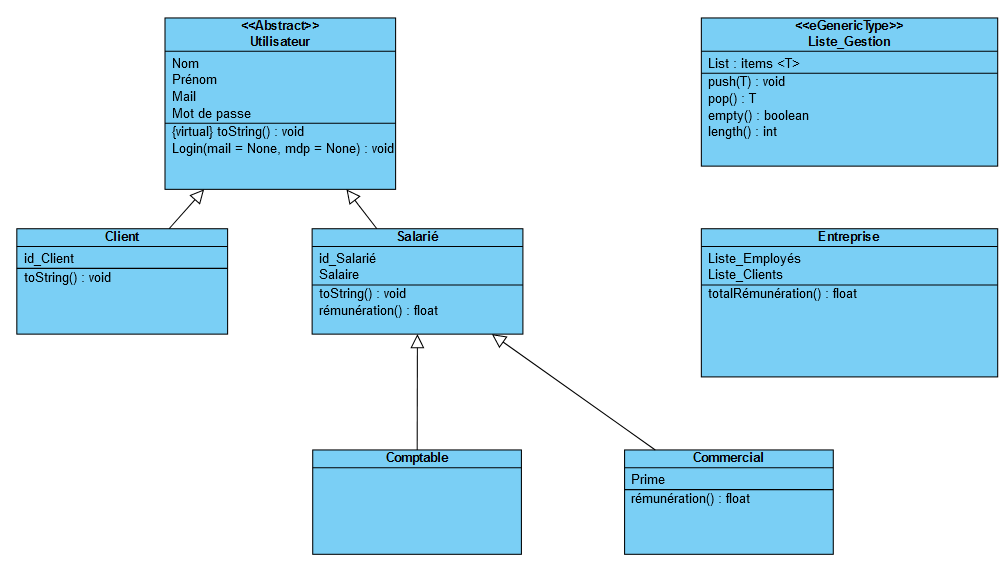
### Cas de figure

Soit un système de gestion d’entreprise simplifié qui comporte un certain nombre de clients et d’employés. On considèrera que cette entreprise est composée d’employés commerciaux et comptables.

Le système devra être capable de créer des clients, des employés (qu’ils soient comptables, commerciaux ou ayant un statut indéterminé) et de les ajouter à l’entreprise à travers des listes qui seront des attributs de cette classe.

Une fonction toString() sera implémentée afin de connaitre le statut de l’instance créée, de plus l’entreprise devra être capable de calculer la somme des rémunérations de tous les employés de l’entreprise. Le système sera aussi capable de vérifier les identifiants de connexion rentrés par un utilisateur.

### Modélisation Objet



A travers cette modélisation nous pourrons mettre en œuvre les concepts clés énoncés précédemment avec notamment :

* Le Polymorphisme : qui fait référence à la capacité d'une variable, d'une fonction ou d'un objet à prendre plusieurs formes, c'est-à-dire à sa capacité de posséder plusieurs définitions différentes
  + La fonction toString() d’équivalent « virutal » pour du C++ par exemple ne renverra aucune action : l’utilisateur dérivera donc de la classe abc qui définit des classe abstraites
  + toString() sera donc obligatoirement définie dans les classes Client et Salarié
* L’overriding qui permettra de redéfinir d’une façon différente une fonction d’une classe fille présente dans une classe mère
  + Il sera illustré à travers la redéfinition de la fonction rémunération () :
    - Rémunération() est définie dans la classe Salarié : cette fonction renverra le salaire mensuel d’une personne
    - Rémunération() est redéfinie dans la classe Commercial qui hérite elle-même de Salarié : cette fonction renverra le salaire mensuel d’une personne en prenant en compte les primes mensuelles du commercial
* L’overloading permettra quant à lui de redéfinir une même méthode dans une même classe avec des paramètres différents :
  + Il sera illustré à travers la fonction Login() qui pourra être définie avec les paramètres :
    - Mail : adresse mail de l’utilisateur
    - Mdp : mot de passe de l’utilisateur
  + Si la méthode reçoit 2 paramètres corrects alors il sera connecté
  + Si la méthode 2 mauvais paramètres alors il ne sera pas connecté
  + Si la méthode reçoit 1 paramètre ou aucun des deux alors il ne sera pas connecté
* La Généricité consiste à définir des algorithmes identiques opérant sur des données de types différents. On définit de cette façon des procédures ou des types entiers génériques :
  + Il sera illustré à travers la classe Liste\_Gestion qui fera dériver des types génériques T.
    - A travers cette classe on pourra donc gérer des listes d'un Type T à instancier et ainsi surcharger les fonctions qui sont utiles à la gestion que l’on prévoit d’effectuer sur nos employés.
  + La généricité sera ensuite utilisée dans la classe Entreprise puisque ses attributs seront à la fois des listes de clients et des listes d’employés.
    - On cherchera donc à uniformiser la gestion de ses deux attributs
* La modularité :

### Analyse de l’implémentation

Après avoir commenté les différents aspects conceptuels et dans quelle partie du programme ces concepts trouvent leur place dans le code, nous allons analyser la manière dont ils sont implémentés et les avantages/inconvénients qu’ils présentent par rapport à des langages objets classiques du type C++.

#### Polymorphisme

**toString():**

Une méthode est dite **virtuelle** si, lorsqu'elle est redéfinie dans une classe dérivée, le type d'instance (classe fille ou mère) détermine la méthode réelle à appeler lors de l'exécution (permet le **polymorphisme**)

Dans l’implémentation présente nous souhaitons que **toString()** s’adapte au statut de l’utilisateur de l’entreprise : Client ou Employé

Nous définissons donc cette fonction dans la classe utilisateur :

Ici, nous utilisons un nouveau mot clef **pass** qui sert à créer une fonction vide. En effet, le mot clef **pass** ne fait strictement rien en Python. On est obligés de l’utiliser pour créer une fonction vide car si rien n’est écrit dans notre fonction l’interpréteur Python va renvoyer une erreur.

Ma classe utilisateur disposera donc d’une fonction toString qui ne contient pas d’instructions.

Dans la mesure où cette fonction ne contient pas de d’instructions lui étant propre il conviendra d’y rajouter le décorateur **ABC** de la metaclass **abc** qui définira le fait qu’aucune classe **Utilisateur** ne pourra être instanciée de plus, aucune des classes dérivées ne pourront l’être tant que la méthode **toString()** ne sera pas redéfinie dans les classes dérivant de **Utilisateur** : à savoir **Client** et **Employé**

#### Overriding

**Remuneration() :**

L’overriding se présente de la même manière que dans les programmes orientés objet classiques.

La classe parent et enfant devront avoir la même signature. Ils devront avoir le même nombre de paramètres. De plus, le type renvoyé pourra différer entre les 2 définitions mère/fille. Il permettra à la classe fille d’adapter grandement l’exécution de toute méthode définie dans la classe mère.

En revanche il faut préciser que la différence majeure de python est qu’il est un langage interprété et non compilé. L’interpréteur ira donc chercher la définition la plus proche : c’est-à-dire celle l’instance de classe fille en question.

NB : Pas d’overriding admit pour les fonctions du type **static/private/final.**

#### Overloading

**Login():**

La surcharge de méthodes ou overloading est un sous partie du polymorphisme. Il représente le procédé par lequel un même « titre » de fonction peut être redéfini plusieurs fois dans une même classe.

Python ne supporte pas l’overloading : plusieurs redéfinitions d’une même fonction différent selon les arguments passés en en-tête.

Python renverra une erreur à l’interprétation si on effectue une implémentation du type C++/JAVA. Ceci s’explique par le fait que python ne soit pas un langage typé ou plutôt « auto-typé ».

Comme j’ai pu le vérifier d’après la source suivante : <https://www.scaler.com/topics/function-overloading-in-python/>

Une image contenant texte

Description générée automatiquement

Cette implémentation renverra une erreur car python prendra seulement en compte la dernière définition de cette méthode : c-a-d la définition à 3 paramètres.

En revanche il est possible d’implémenter cette caractéristique dans un programme Orienté Objet en python :

Il suffira d’envoyer **None** sur un attribut ou plus passés en argument de notre méthode. La manière dont procèdera cette méthode diffèrera selon les cas en regardants quels attributs sont envoyés en argument.

NB : Les méthodes du type **static/private/final** pourront être ‘overloadées’

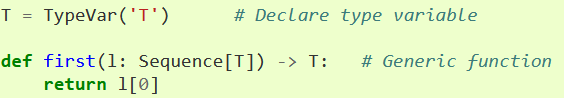
#### Généricité

**Liste\_Gestion :**

En python la généricité trouve son sens au travers de l’import du module typing intégré à python et de ses factories TypeVar et Generic.

TypeVar permettra de déclarer un type de variable, ils servent de paramètres pour les classes et fonctions génériques.

Comme évoqué plus haut : le factory Generic nous permettra de définir des fonctions génériques :



Ainsi que des classes génériques :

Une image contenant texte

Description générée automatiquement

Dans le cas des classes génériques (implémentées dans notre code) le type ‘T’ sera donc rendu valide en tant que paramètre ainsi qu’en type de retour et dans le corps de nos méthodes : le type est reconnu.

Source : <https://docs.python.org/3/library/typing.html>

La classe Liste\_Gestion nous permettra de gérer des listes d’un type souhaité : dans le cas présent des listes de clients et d’employés.

#### Modularité

## Analyse de données

Dans ce TP il nous est aussi demandé de montrer les avantages de Python dans les analyses de données avec une étude de cas.

Nous tenterons donc à travers une étude de cas que nous avons imaginé de présenter la manière d’introduire ces concepts de programmation.

### Cas de figure

Dans la mesure ou l’un d’entre nous a actuellement des cours d’apprentissage automatique avec des activités d’analyse de données, il nous a semblé pertinent de reprendre ce cas de figure en essayant d’édulcorer au maximum les dépendances au problème tout en montrant les concepts intéressants autour de la programmation python en analyse.

Ceci a aussi permis de rendre le code de génération de données (auparavant procédural) un peu plus OO pour rester dans le cadre du cours. La création de certaines classes rend forcément le code plus structuré donc plus lisible.

Le problème dans lequel on se place est un problème de résolution par **éléments finis :** la méthode des éléments finis est utilisée pour résoudre numériquement des équations aux dérivées partielles.

Cette méthode permet notamment de discrétiser numériquement des structures physiques et d’extraire des données d’intérêt.

On se placera physiquement dans le cas d’une poutre encastrée en l’une de ses extrémités et libre sur l’autre avec un poids P appliquée sur cette extrémité dont voici le schéma :

Une image contenant texte, antenne

Description générée automatiquement

Cette poutre réagira au poids appliqué en résonnant à certaines fréquences (en réalité une infinité).

Notre but sera donc de modéliser un dataset d’entrées sous forme de csv et d’en extraire un autre dataset de sortie sous forme csv au travers d’une résolution par éléments finis sur une poutre encastrée libre. Nous effectuerons ensuite une analyse de données sur ces 2 csv.

#### Entrées

Les entrées du problème seront les caractéristiques physiques des différentes instances de poutres générées avec 10 entrées :

* NbElts
* L\_tot
* Rho
* H
* B
* S
* I
* L
* E
* Mat

Ces caractéristiques résultent de la modélisation physique du code ‘generate\_data\_learn\_V1\_10elts.py’.

Au niveau de l’analyse de données nous nous intéresserons notamment aux matériaux ‘Mat’ et notamment des matériaux en ‘Aluminium’.

#### Sorties

Les sorties du problème seront les caractéristiques de fréquence des différentes instances de poutres générées avec 8 fréquences dites de résonnance :

* Freq1
* Freq2
* Freq3
* Freq4
* Freq5
* Freq6
* Freq7
* Freq8

Ces caractéristiques résultent de la modélisation physique du code ‘generate\_data\_learn\_V1\_10elts.py’.

Au niveau de l’analyse de données nous nous intéresserons aux fréquences ‘Freq1’ et notamment des instances dont les matériaux correspondent à de l’aluminium.

#### Objectif de l’analyse de données

A travers cette analyse de données nous explorerons :

* Les avantages et les facilités du langage python vis-à-vis de l’analyse de données
* Les avantages et les facilités offertes par les librairies développées par la communauté vis-à-vis de l’analyse de données

### Analyse de l’implémentation

#### Pandas & Seaborn

#### Numpy & Listes

## Conclusion