Intitulé du cours : Programmation orientée objets

Enseignant : M HENDRIGUE NKOLO Joseph

Objectifs du cours : Comprendre les paradigmes de la programmation orientée objet (POO) ;

Chapitre 1 : Introduction aux concepts orientés objets

Historiquement il existe deux grandes approches de la programmation informatique : l'approche procédurale et l'approche orientée objet. La programmation orientée objet (POO) est un paradigme ou une approche de programmation élaboré par les norvégiens Ole-Johan Dahl et Kristen Nygaard au début de la décennie 1960, puis par l'américain Alan Kay dans les années 1970. Avant que la POO ne soit employée, la programmation informatique se faisait au moyen de la programmation procédurale.

La POO ne permet pas en principe de résoudre plus de problèmes ou de développer plus de fonctionnalités que la programmation procédurale. Elle permet toutefois de mieux organiser le code applicatif. Cette meilleure structuration du code devrait faciliter le développement, la maintenance, la réutilisation et l'évolutivité des programmes.

L'objectif principale de ce chapitre est de montrer ces avantages de la POO par une comparaison avec la programmation procédurale. Cette comparaison nous permettra d'introduire l'approche orientée objet.

1.1. Comparaison entre la programmation procédurale et la programmation orienté objet

Dans la programmation procédurale, la résolution d'un problème est réalisée en décomposant ce problème en sous-problèmes. Ainsi, le programme principal est décomposé en procédures (ou fonctions) qui résolvent chacune un sous-problème et interagissent entre elles afin de résoudre le problème principal.

Dans cette approche de programmation, le programme principal est généralement une procédure. Les programmes dans une approche procédurale forment alors des « hiérarchies » de procédures ou fonctions.

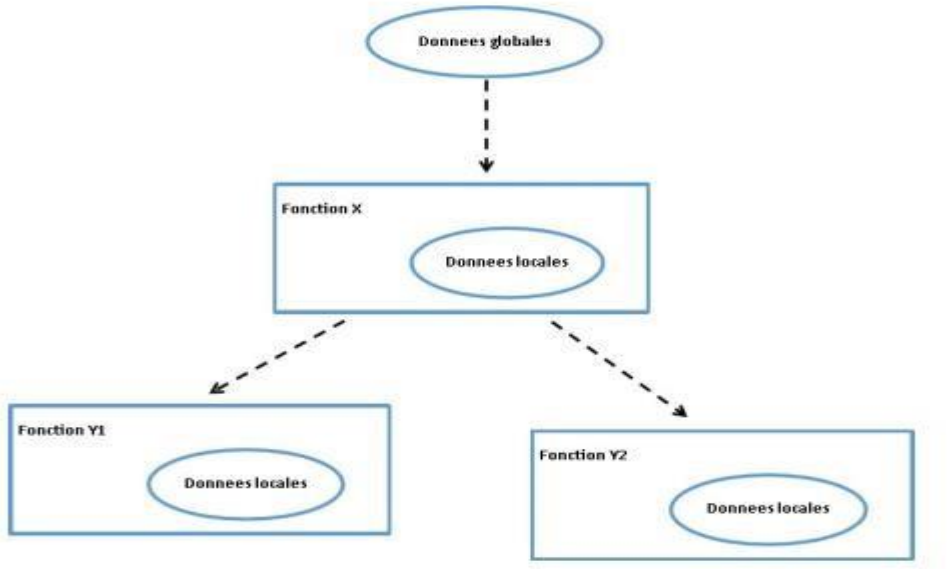

Figure 1: Architecture d'un programme procédural

Comme avantages, et limites de la programmation procédurale nous avons :

antages	Limites	Quand utiliser la Programmatio procédurale ?
• Simplicité : Concept facile à comprendre, surtout pour les débutants. Structure linéaire qui suit le flux logique d'un algorithme. • Efficacité : Souvent plus rapide en termes d'exécution que d'autres paradigmes, en particulier pour des tâches simples.	• Complexité croissante : Pour des programmes de grande taille, la gestion du code peut devenir complexe • Manque de modularité : Bien que les procédures permettent une certaine modularité • Difficulté à modéliser des problèmes complexes : Les problèmes du monde réel, souvent complexes et avec des relations non linéaires • Moins adaptée à l'évolution : Les modifications apportées à	• Petits programmes • Tâches spécifiques • Performances critiques : Lorsque la vitesse d'exécution est primordiale • Langages de bas niveau

	un programme peuvent avoir des conséquences inattendues sur d'autres parties du code, rendant la maintenance plus délicate.	

A contrario, la programmation par objet est un paradigme qui consiste à identifier, définir et assembler les idées, concepts ou entités nécessaire à la résolution d'un problème. Ainsi, le programme principal est un assemblage « briques logicielles » appelées objets. Un objet représente alors un concept, une idée ou une entité utilise pour la résolution d'un problème. Les programmes dans cette forme alors des « réseaux » d'objets qui interagisse.

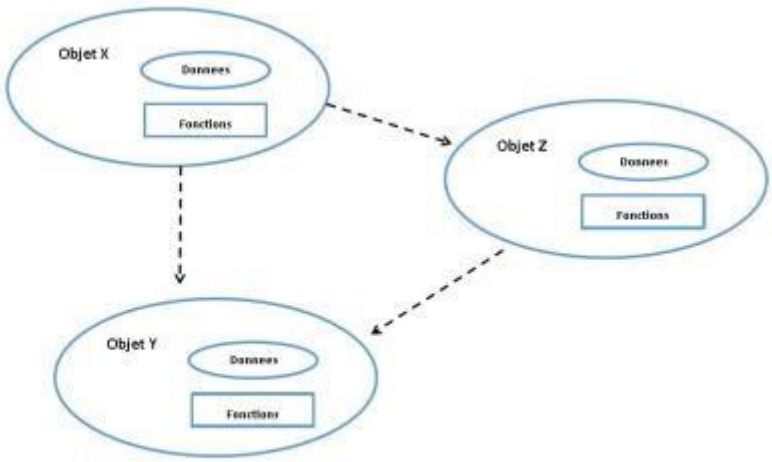

Figure 2: Architecture d'un programme orienté objet

Si la programmation procédurale est intuitive lorsqu'il s'agit d'apprendre la programmation, cette méthode pose un certain nombre d'inconvénients à terme. Le premier est que la plus petite modification de la structure des données du programme principal appelle à une modification de toutes les procédures qui interagissent avec ces données. De plus, développer et maintenir un très grand programme en procédural peut s'avérer long et fastidieux. Comme avantages, et limites de la programmation procédurale nous avons :

antages	Limites	Quand utiliser la Programmation procédurale ?
• La modularité : Grâce à la POO, on peut former des modules	• Peu intuitive : Elle sera moins facile d'accès que	• Les grands projets complexes

• compacts regroupant des données et un ensemble d'opérations. L'abstraction : La POO permet de créer des objets abstraits à partir du réel. Ses concepts se rapprochent donc des • fonctionnalités utilisées par tout un chacun au quotidien.. La maintenance et l'évolutivité : modifier un des constituants du programme n'affectera pas les • autres et n'entraînera donc pas d'erreurs Le développement et la réutilisation : l'indépendance des objets permet de les réutiliser dans d'autres applications ou programmes.	• l'approche procédurale, généralement la première à être apprise lorsqu'on débute, ou l'approche fonctionnelle qui parlera tout de suite aux matheux. Exigeante : La POO demande une grande rigueur dans le code pour que ses concepts s'appliquent correctement.	• Les applications graphiques • Les simulations • Les systèmes distribués

Les différences entre la POO et la programmation procédurale sont présentées dans le tableau suivant :

POO	Programmation Procédurale
Concept central : Le programme est organisé autour d'objets qui représentent des entités du monde réel (personnes, objets, concepts).	**Concept central** : Le programme est divisé en une série d'instructions qui s'exécutent séquentiellement.
Focus: Sur les objets et leurs interactions.	**Focus**: Sur les actions à effectuer (les procédures ou fonctions).
Structure: Hiérarchique, avec des classes (modèles d'objets) et des objets (instances de classes).	Structure: Linéaire, avec un flux de contrôle bien défini.
Données et fonctions: Encapsulées dans les objets (attributs et méthodes).	**Données**: Considérées comme des données séparées des fonctions qui les manipulent.

Exemple: Un programme simulant un jeu vidéo où chaque personnage est un objet avec ses propres attributs (vie, force) et méthodes (attaquer, se déplacer)	**Exemple**: Un programme qui calcule une moyenne en suivant étape par étape les opérations arithmétiques.

1.2. Notion de Classe et d'Objet

Une classe est un modèle définissant des caractéristiques (attributs) et des actions (méthodes) communes aux objets qui en sont dérivés. Le concept de classe devient beaucoup plus clair avec un exemple :

Exemple : Imaginons une classe Voiture. Cette classe contiendrait des attributs comme la marque, le modèle, et la couleur, et des méthodes comme démarrer et arrêter qui permettent de démarrer et d'arrêter la voiture, respectivement

Un objet est une instance spécifique d'une classe, contenant des valeurs réelles pour les attributs définis par la classe et pouvant exécuter les méthodes qui y sont spécifiées. C'est comme un exemplaire individuel créé à partir du plan fourni par la classe.

Exemple : À partir de la classe Voiture, nous pouvons créer des objets spécifiques, comme ma_voiture = Voiture ("Toyota", "Corolla", "rouge"). Ici ma_voiture est un objet avec ses propres valeurs pour les attributs marque, modèle, et couleur.

II. Les principes de l'orienté Objet

La POO repose sur 4 principes : L'encapsulation ; L'abstraction ; L'héritage ; Le polymorphisme.

2.1. L'encapsulation

L'encapsulation consiste à regrouper les données (attributs) et les méthodes (fonctions) qui agissent sur ces données au sein d'une même entité : l'objet. C'est comme créer une "boîte noire" où l'on met tout ce qui concerne un objet particulier.

2.1.1. Avantages de l'encapsulation :

- Protection des données : Les données internes d'un objet sont protégées des accès non autorisés, ce qui garantit leur intégrité.
- Modularité : Chaque objet est une unité indépendante, ce qui facilite la compréhension et la maintenance du code.

- Réutilisabilité : Les objets encapsulés peuvent être réutilisés dans différents contextes.

Exemple :

Prenons l'exemple d'une classe Voiture. Les attributs pourraient être marque, modèle, couleur, vitesseMax, etc. Les méthodes pourraient être accelerer(), freiner(), tourner(), etc. En encapsulant ces éléments dans la classe Voiture, on crée une abstraction qui représente une voiture réelle.

2.1.2. Encapsulation et Masquage d'information

L'encapsulation et le masquage sont étroitement liés. En encapsulant les données et les méthodes dans un objet, on crée une frontière entre l'intérieur et l'extérieur de cet objet. Le masquage permet alors de définir les règles d'accès à cette frontière. Le masquage est un mécanisme qui permet de contrôler l'accès aux attributs et aux méthodes d'un objet. On distingue généralement trois niveaux d'accès :

- Public : Accessible de partout.
- Protected : Accessible uniquement par les méthodes de la classe et de ses sous-classes.
- Private : Accessible uniquement par les méthodes de la classe.

Exemple : Dans la classe Voiture, l'attribut vitesseMax pourrait être déclaré comme private pour empêcher une modification directe de sa valeur. Pour modifier la vitesse maximale, on utiliserait une méthode publique comme setVitesseMax()

2.2. L'abstraction

L'abstraction est un concept fondamental en programmation orientée objet (POO) qui consiste à simplifier la complexité d'un système en se concentrant sur les aspects essentiels et en ignorant les détails d'implémentation. C'est comme créer un modèle simplifié d'un objet réel, en ne retenant que les caractéristiques qui nous intéressent.

Les Classes abstraites sont des classes qui ne peuvent pas être instanciées directement. Elles servent de modèle pour d'autres classes et définissent les méthodes que les classes filles doivent implémenter.

Une interface définit un contrat : elle spécifie les méthodes que les classes qui implémentent cette interface doivent avoir, sans se soucier de leur implémentation.

Exemple :

Classe abstraite Véhicule : Elle pourrait définir des méthodes abstraites comme accelerer (), freiner(), tourner().

Classes filles : Voiture, Moto, Camion qui héritent de Véhicule et implémentent les méthodes de manière spécifique à chaque type de véhicule.

2.3. L'Héritage

L'héritage est un mécanisme fondamental en programmation orientée objet (POO) qui permet de créer de nouvelles classes en réutilisant les propriétés et les méthodes d'une classe existante. C'est un peu comme une relation de parenté entre les classes : une classe fille hérite des caractéristiques de sa classe mère.

2.3.1. Avantages de l'héritage

- Réutilisation de code : Évitez de réécrire du code en réutilisant les propriétés et les méthodes d'une classe existante.
- Hiérarchie de classes : Organisez votre code de manière hiérarchique, en créant des relations "est-un" entre les classes.
- Polymorphisme : Permet de traiter des objets de différentes classes de manière uniforme.
- Organisation du code : Crée une structure hiérarchique claire.
- Réduction de la redondance : Évite de réécrire du code identique.
- Extensibilité : Permet de créer de nouvelles classes en spécialisant les classes existantes.

En héritage nous avons les concepts suivants :

- Classe mère (superclasse): La classe dont on hérite.
- Classe fille (sous-classe): La nouvelle classe qui hérite de la classe mère.
- Héritage : La classe fille possède toutes les propriétés et méthodes de la classe mère, en plus de celles qu'elle définit elle-même.

2.4. Le Polymorphisme

Le polymorphisme est un concept clé en programmation orientée objet (POO) qui confère aux objets la capacité de prendre plusieurs formes. En d'autres termes, une même méthode peut avoir des comportements différents lorsqu'elle est appelée sur des objets de classes différentes mais liées par une relation d'héritage. Il existe deux principaux types de polymorphisme :

- Polymorphisme par sous-typage (ou par redéfinition) : Une méthode d'une classe fille redéfinit (override) une méthode héritée de la classe mère.
- Polymorphisme ad hoc (ou par surcharge) : Plusieurs méthodes portent le même nom mais ont des signatures différentes (nombre ou type de paramètres).

CHAPITRE 2 : POO en PYTHON