PROGRAMMATION ORIENTE OBJETS (POO)

Pour chaque concept vous donnerez un code python pour l'illustrer

- Paradigme de programmation

Un paradigme de programmation est une façon de penser et de structurer un programme informatique. Python est un langage de programmation multi-paradigme, ce qui signifie qu'il permet d'utiliser différents styles de programmation pour résoudre un problème.

- Programmation procédurale

La programmation impérative est le paradigme de programmation le plus ancien. Il consiste à résoudre un problème en listant une série d'instructions (assignations, boucles, conditions, opérations) qui modifient l'état d'un programme jusqu'à obtenir la solution.

- Programmation orienté objets

La POO est un paradigme de programmation qui permet aux programmeurs de modéliser les objets du monde réel et les interactions entre eux à l'aide de classes et d'objets.

- Namespace

Le "Namespace" ou "espace de noms" est une structure de données qui permet de stocker les noms de variables, de fonctions et de classes dans un programme. Cette structure permet d'organiser et de gérer les noms de manière à éviter les conflits de noms et à améliorer la lisibilité et la maintenance du code.

- Classe

En Python, une classe est une structure de données qui permet de regrouper des données et des fonctions associées dans un objet. Elle définit un ensemble d'attributs (variables) et de méthodes (fonctions) qui définissent le comportement de l'objet. Les classes permettent de créer des types de données personnalisés qui peuvent être utilisés dans un programme.

```
class Personne:
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age
    def parler(self, message):
        print(self.nom, ":", message)
```

Dans cet exemple, nous avons créé une classe Personne qui a deux paramètres(attribut) : nom et âge. La méthode spéciale __init__ est un constructeur qui initialise les attributs de la classe lors de sa création. La méthode parler est une méthode de la classe qui prend un paramètre message et affiche le nom de la personne suivi du message.

En résumé, les classes en Python sont un outil puissant pour créer des types de données personnalisés qui peuvent être utilisés dans un programme. Elles permettent de regrouper des données et des fonctions associées dans un objet et de définir le comportement de l'objet en définissant ses attributs et ses méthodes.

- ✓ Concrete

Une classe concrète est une classe qui peut être instanciée et utilisée pour créer des objets. En d'autres termes, une classe concrète est une classe qui a une implémentation complète de toutes ses méthodes et qui peut être utilisée pour créer des objets réels.

```
class Rectangle:
    def __init__(self, largeur, hauteur):
        self.largeur = largeur
        self.hauteur = hauteur
    def aire(self):
        return self.largeur * self.hauteur
    def perimetre(self):
        return 2 * (self.largeur + self.hauteur)
```

- ✓ Abstraite

Une classe abstraite est une classe qui ne peut pas être instanciée directement, mais qui sert de modèle pour d'autres classes qui en héritent. Elle définit des méthodes et des attributs communs qui doivent être implémentés dans les sous-classes pour être utilisables. Les classes abstraites sont souvent utilisées pour créer une hiérarchie de classes qui ont des fonctionnalités communes.

```
from abc import ABC, abstractmethod

class Figure(ABC):
    @abstractmethod
    def aire(self):
        pass
    @abstractmethod
    def perimetre(self):
        Pass
```

- ✓ Attributs

Un attribut est une variable qui est associée à une instance ou à une classe. Les attributs sont utilisés pour stocker des données spécifiques à une instance ou à une classe, et ils peuvent être accédés et modifiés à travers les méthodes de la classe.

- Instance

Les attributs d'instance en Python sont des variables qui sont associées à une instance particulière d'une classe. Chaque instance d'une classe peut avoir ses propres valeurs pour les attributs d'instance, ce qui signifie que chaque instance peut avoir des valeurs différentes pour les mêmes attributs.

```
class Personne:
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age
```

Dans cet exemple, nous avons une classe Personne qui a deux attributs d'instance : nom et age. Ces attributs sont initialisés avec des valeurs spécifiques pour chaque instance de la classe lorsqu'elles sont créées.

- Classe

Les attributs de classe sont des variables qui sont associées à une classe plutôt qu'à une instance de la classe. Cela signifie que l'attribut de classe est partagé par toutes les instances de la classe et peut être accédé directement à partir de la classe elle-même.

```
class Personne:
    # Attribut de classe
    nb_personnes = 0
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age
        # On incrémente le compteur d'instances à chaque création d'une nouvelle instance de la classe Personne
        Personne.nb_personnes += 1
```

Dans cet exemple, nous avons une classe Personne qui a un attribut de classe nb_personnes. Cet attribut est initialisé à zéro et est partagé par toutes les instances de la classe. Nous avons également une méthode __init__ qui est appelée à chaque création d'une nouvelle instance de la classe. Cette méthode incrémente le compteur d'instances nb_personnes à chaque création d'une nouvelle instance.

- ✓ Methodes

les méthodes sont des fonctions associées à une classe ou à une instance de classe. Les méthodes de classe et les méthodes d'instance sont les deux types de méthodes que l'on peut définir dans une classe.

- Instances

Les méthodes d'instance sont des fonctions qui peuvent accéder aux attributs d'instance d'une classe et effectuer des opérations sur ces attributs. Pour définir une méthode d'instance dans une classe, on utilise le mot-clé def suivi du nom de la méthode et des paramètres éventuels. Par convention, le premier paramètre d'une méthode d'instance est self, qui fait référence à l'instance actuelle de la classe.

```
class Personne:
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age
    def afficher_informations(self):
        print(f"Nom : {self.nom}, Age : {self.age}")
```

Dans cet exemple, nous avons une classe Personne avec une méthode d'instance afficher_informations(). Cette méthode affiche les valeurs des attributs d'instance nom et age de l'instance actuelle de la classe.

- Static

les méthodes statiques (ou méthodes de classe statique) sont des méthodes qui peuvent être appelées directement sur la classe sans instancier celle-ci, et qui n'ont pas besoin d'accéder à l'état interne d'une instance ou d'une classe

```
class Calculateur:
    @staticmethod
    def multiplier(a, b):
        return a * b
```

Dans cet exemple, nous avons une classe Calculateur avec une méthode statique multiplier (). Cette méthode prend deux paramètres a et b et renvoie leur produit.

- Constructeurs

, le constructeur est une méthode spéciale appelée __init__() qui est exécutée automatiquement lorsqu'une instance d'une classe est créée. Le constructeur est utilisé pour initialiser les attributs d'une instance de la classe avec des valeurs spécifiques, et peut également effectuer d'autres opérations nécessaires à l'initialisation de l'objet.

- ✓ Encapsulation

L'encapsulation en Python permet de protéger les données et les méthodes d'une classe en limitant leur accès depuis l'extérieur. Cela permet de prévenir toute modification non autorisée et de maintenir une meilleure structure de code.

- Visibilité ou Portée d'un attribut ou d'une Méthode

La visibilité ou la portée d'un attribut ou d'une méthode détermine l'endroit où il peut être accédé dans le code. En Python, la portée d'un attribut ou d'une méthode peut être définie à l'aide de l'encapsulation.

- Getters

Les getters sont des méthodes utilisées pour accéder aux attributs encapsulés d'une classe. Ils sont utilisés pour récupérer la valeur d'un attribut encapsulé, tout en contrôlant l'accès à l'attribut.

- Setters

Les setters sont des méthodes utilisées pour modifier la valeur d'un attribut encapsulé d'une classe. Ils sont utilisés pour définir la valeur d'un attribut encapsulé, tout en contrôlant l'accès à l'attribut.

- ✓ Surcharge

la surcharge (overloading) de méthode telle qu'on la trouve dans d'autres langages de programmation orientée objet comme Java ou C++ n'est pas possible. Cela est dû à la façon dont Python traite les arguments de méthode et leur typage.

Cependant, il est possible de simuler la surcharge de méthode en Python en utilisant des arguments par défaut et des paramètres optionnels

- ➢ Objets

un objet est une instance d'une classe qui peut contenir des attributs (variables) et des méthodes (fonctions) associées.

- ➢ Relation entre Classe

il est courant d'avoir des relations entre classes. Les deux relations les plus courantes sont l'héritage et la composition.

L'héritage permet à une classe d'hériter des propriétés et des comportements d'une autre classe, appelée classe parent ou superclasse. La classe héritante est appelée sous-classe ou classe enfant.

- ✓ Navigabilité entre Classe

La navigabilité entre les classes en Python se réfère à la capacité des instances d'une classe à interagir avec les instances d'autres classes. Cela peut être réalisé par composition, héritage, références ou messages

- OneToMany

Il s'agit d'une relation où un objet de la classe parent peut être associé à plusieurs objets de la classe enfant, mais chaque objet enfant ne peut être associé qu'à un seul objet parent.

- ManyToOne

Il s'agit d'une relation où plusieurs objets de la classe enfant peuvent être associés à un seul objet de la classe parent.

- ManyToMany

il s'agit d'une relation où chaque objet de la classe parent peut être associé à plusieurs objets de la classe enfant et vice versa.

- OneTone

il s'agit d'une relation où chaque objet de la classe parent est associé à un seul objet de la classe enfant et vice versa.

- ✓ Heritage

Un concept cle de la poo il permet à une classe de dériver les attributs et les méthodes d'une classe parente. En Python, cela se fait en utilisant le mot clé class suivi du nom de la classe dérivée, suivi du nom de la classe parente entre parenthèses.

- Redéfinition

La redéfinition est la capacité d'une classe dérivée de remplacer une méthode de sa classe parente par sa propre version. En d'autres termes, la méthode de la classe parente est "redéfinie" dans la classe dérivée. Cela permet à la classe dérivée de personnaliser ou de modifier le comportement de la méthode héritée pour répondre à ses besoins spécifiques

- Polymorphisme

Le polymorphisme est un concept qui permet aux objets d'une même hiérarchie de classe de répondre différemment à la même méthode. Cela signifie que des objets différents peuvent être traités de la même manière

- ➢ Interface

En Python, il n'y a pas de syntaxe intégrée pour définir des interfaces. Cependant, on peut créer une interface en Python en utilisant une classe abstraite. Pour cela, on définit une classe abstraite avec les méthodes abstraites que les classes dérivées doivent implémenter. Les classes dérivées peuvent alors implémenter les méthodes de l'interface selon les besoins. Une classe qui contient des méthodes abstraites ne peut pas être instanciée et doit être dérivée pour fournir des implémentations concrètes des méthodes abstraites.