PYTHON

Présenté par :

Xavier TABUTEAU

Polymorphisme - Encapsulation

Les classes avancées

- **Polymorphisme**

Le polymorphisme est un mécanisme important dans la programmation objet. Il permet de modifier le comportement d'une classe fille par rapport à sa classe mère. Le polymorphisme permet d'utiliser l'héritage comme un mécanisme d'extension en adaptant le comportement des objets.

- **Propriétés**

Les propriétés permettent de définir des comportements de 'getter' et 'setter' sur les méthodes d'une classe. Cela nous permet également d'appeler une méthode sans avoir besoin d'utiliser les parenthèses.

- Pour créer une propriété de type getter, on utilise le décorateur @property sur une méthode.
- Pour rajouter un 'setter', il faut décorer la méthode du même nom avec un décorateur ayant la syntaxe @nom_method.setter. Si aucun setter n'est défini alors elle n'est pas modifiable, par conséquent c'est une constante.
- Il existe aussi un 'deleter' qui permet de supprimer la propriété. La syntaxe est la même que le setter en remplaçant « setter » par « deleter ».

polymorphisme.py

Polymorphisme - Encapsulation

Les classes avancées

- **Encapsulation**

En Python, il n'existe pas de mécanisme dans le langage qui nous permettrait de gérer la visibilité. Par contre, il existe une convention dans le nommage. Une méthode ou un attribut dont le nom commence par « _ » (underscore) est considéré comme privé. Il est donc déconseillé d'accéder à un tel attribut ou d'appeler une telle méthode depuis l'extérieur de l'objet.

- **Masquer des attributs / méthodes à une classe fille**

Parfois, on ne souhaite pas qu'une méthode puisse être redéfinie ou qu'un attribut puisse être modifié dans une classe fille. Pour cela, il suffit que le nom de la méthode ou de l'attribut commence par deux « _ ». Nous avons vu précédemment que Python n'a pas de mécanisme pour contrôler la visibilité des éléments d'une classe. Par convention, les développeurs signalent par un « _ » le statut privé d'un attribut ou d'une méthode. Par contre le recours à deux underscores à un impact sur l'interpréteur qui renomme la méthode ou l'attribut sous la forme :

  - _<nom de la classe>__<nom>

propriete_encapsulation.py

Polymorphisme - Encapsulation

Les classes avancées

- **Destructeur**

Les destructeurs sont appelés lorsqu'un objet est détruit. En Python, les destructeurs ne sont pas aussi nécessaires que en C++, car Python dispose d'un ramasse-miettes qui gère automatiquement la gestion de la mémoire.

La méthode __del __() est une méthode appelée destructeur en Python. Elle est appelée lorsque toutes les références à l'objet ont été supprimées, c'est-à-dire lorsqu'un objet est nettoyé via le mot clé « del » ou à la fin du programme par exemple.

- **Fermer la liste des attributs**

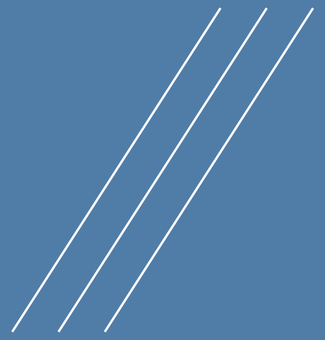
Il est possible déclarer la liste finie des attributs. Pour cela, on déclare un attribut de classe appelé __slots__.

Lorsque __slots__ n'est pas indiqué, on peut ajouter des attributs inexistants à la création d'une classe. Si on l'indique, cela n'est plus possible et génère une erreur de type AttributeError.

L'attribut __slots__ a également une utilité pour l'optimisation du code. Comme on indique à l'interpréteur le nombre exact d'attributs, il peut optimiser l'allocation mémoire pour un objet de ce type.

destructeur.py
ex_propriete_encapsulation

PYTHON



Présenté par

Xavier TABUTEAU