**9. Classes**

**Les classes permettent de regrouper des données et des fonctionnalités.**

La création d'une nouvelle classe crée un nouveau type d'objet et permet de créer de nouvelles instances de ce type.

Chaque instance de classe peut avoir des attributs qui lui sont attachés pour maintenir son état.

Les instances de classe peuvent également avoir des méthodes (définies par leur classe) pour modifier leur état.

Par rapport à d'autres langages de programmation, le mécanisme de classe de Python ajoute des classes avec un minimum de nouvelle syntaxe et sémantique.

Il s'agit d'un mélange des mécanismes de classe que l'on trouve en C++ et en Modula-3. Les classes Python offrent toutes les caractéristiques standard de la programmation orientée objet :

le mécanisme d'héritage des classes autorise plusieurs classes de base, une classe dérivée peut remplacer n'importe quelle méthode de sa ou ses classes de base, et une méthode peut appeler la méthode d'une classe de base portant le même nom. Les objets peuvent contenir des quantités et des types de données arbitraires.

Comme c'est le cas pour les modules, les classes participent à la nature dynamique de Python : elles sont créées au moment de l'exécution et peuvent être modifiées après leur création.

Dans la terminologie C++, les membres de la classe (y compris les membres données) sont normalement publics (sauf voir ci-dessous Variables privées), et toutes les fonctions membres sont virtuelles. Comme dans Modula-3, il n'y a pas de raccourcis pour référencer les membres de l'objet à partir de ses méthodes : **la fonction de la méthode est déclarée avec un premier argument explicite représentant l'objet, qui est fourni implicitement par l'appel**. Comme dans Smalltalk, les classes elles-mêmes sont des objets. Cela fournit une sémantique pour l'importation et le renommage. Contrairement à C++ et Modula-3, les types intégrés peuvent être utilisés comme classes de base pour une extension par l'utilisateur. De même, comme en C++, la plupart des opérateurs intégrés avec une syntaxe spéciale (opérateurs arithmétiques, souscriptions, etc.) peuvent être redéfinis pour les instances de classe.

(En l'absence d'une terminologie universellement acceptée pour parler des classes, j'utiliserai occasionnellement des termes de Smalltalk et de C++. J'utiliserais les termes de Modula-3, puisque sa sémantique orientée objet est plus proche de celle de Python que de C++, mais je m'attends à ce que peu de lecteurs en aient entendu parler).

**9.1. Un mot sur les noms et les objets**

Les objets ont une individualité, et plusieurs noms (dans plusieurs portées) peuvent être liés au même objet. Ceci est connu sous le nom d'aliasing dans d'autres langues. Ce phénomène n'est généralement pas apprécié au premier coup d'oeil à Python, et peut être ignoré sans risque lorsqu'il s'agit de types de base immuables (nombres, chaînes de caractères, tuples). Cependant, l'aliasing a un effet surprenant sur la sémantique du code Python impliquant des objets mutables tels que les listes, les dictionnaires et la plupart des autres types. Cet effet est généralement utilisé à l'avantage du programme, puisque les alias se comportent comme des pointeurs à certains égards. Par exemple, le passage d'un objet est peu coûteux puisque seul un pointeur est transmis par l'implémentation ; et si une fonction modifie un objet passé en argument, l'appelant verra le changement - ceci élimine le besoin de deux mécanismes différents de passage d'argument comme en Pascal.

**9.2. Scopes et espaces de noms Python**

Avant de présenter les classes, je dois d'abord vous parler des règles de portée de Python. Les définitions de classes jouent de jolis tours avec les espaces de noms, et vous devez savoir comment fonctionnent les **scopes** et les **espaces** de noms pour bien comprendre ce qui se passe. Incidemment, la connaissance de ce sujet est utile à tout programmeur Python avancé.

Commençons par quelques définitions.

Un espace de noms est une correspondance entre des noms et des objets. La plupart des espaces de noms sont actuellement implémentés sous forme de dictionnaires Python, mais cela n'est normalement pas perceptible de quelque manière que ce soit (sauf pour les performances), et cela pourrait changer à l'avenir.

**Voici quelques exemples d'espaces de noms** : l'ensemble des noms intégrés (contenant des fonctions telles que abs(), et les noms des exceptions intégrées) ; les noms globaux dans un module ; et les noms locaux dans une invocation de fonction.

En un sens, l'ensemble des attributs d'un objet constitue également un espace de noms.

**La chose importante à savoir sur les espaces de noms est qu'il n'y a absolument aucune relation entre les noms dans différents espaces de noms ; par exemple, deux modules différents peuvent tous deux définir une fonction maximize sans confusion - les utilisateurs des modules doivent la préfixer avec le nom du module.**

À propos, j'utilise le mot attribut pour tout nom suivant un point - par exemple, dans l'expression z.real, real est un attribut de l'objet z.

Strictement parlant, les références aux noms dans les modules sont des références à des attributs : **dans l'expression modname.funcname, modname est un objet du module et funcname en est un attribut**. Dans ce cas, il existe une correspondance directe entre les attributs du module et les noms globaux définis dans le module : ils partagent le même espace de noms ! 1

**Les attributs peuvent être en lecture seule ou en écriture.**

**Dans ce dernier cas, l'affectation aux attributs est possible. Les attributs de module sont inscriptibles : vous pouvez écrire modname.the\_answer = 42.**

Les attributs inscriptibles peuvent également être supprimés avec l'instruction **del**.

Par exemple, **del modname.the\_answer** supprimera l'attribut **the\_answer** de l'objet nommé par modname.

Les espaces de noms sont créés à différents moments et ont des durées de vie différentes.

**L'espace de noms contenant les noms intégrés est créé au démarrage de l'interpréteur Python et n'est jamais supprimé.**

L'espace de noms global d'un module est créé lorsque la définition du module est lue ; normalement, les espaces de noms des modules durent également jusqu'à ce que l'interpréteur quitte.

**Les instructions exécutées par l'invocation de haut niveau de l'interpréteur, qu'elles soient lues à partir d'un fichier script ou de manière interactive, sont considérées comme faisant partie d'un module appelé \_\_main\_\_, et ont donc leur propre espace de noms global. (Les noms intégrés vivent en fait également dans un module ; cela s'appelle les builtins).**

**L'espace de noms local d'une fonction est créé lorsque la fonction est appelée, et supprimé lorsque la fonction revient ou lève une exception qui n'est pas traitée dans la fonction.**

(En fait, l'oubli serait une meilleure façon de décrire ce qui se passe réellement). Bien sûr, les invocations récursives ont chacune leur propre espace de noms local.

**Un scope** est une région textuelle d'un programme Python où un espace de noms est directement accessible. "Directement accessible" signifie ici qu'une référence non qualifiée à un nom tente de trouver le nom dans l'espace de noms.

Bien que les scopes soient déterminés statiquement, ils sont utilisés dynamiquement. À tout moment pendant l'exécution,

il existe 3 ou 4 scopes imbriqués dont les espaces de noms sont directement accessibles :

**le scope le plus interne, qui est recherché en premier, contient les noms locaux**

**les scopes de toutes les fonctions englobantes, qui sont recherchés en commençant par le scope le plus proche, contiennent les noms non locaux, mais aussi non globaux**

**l'avant-dernière portée contient les noms globaux du module actuel**

**l'étendue la plus éloignée (recherchée en dernier) est l'espace de noms contenant les noms intégrés.**

Si un nom est déclaré global, alors toutes les références et les affectations vont directement à l'avant-dernière portée contenant les noms globaux du module.

Pour relier des variables situées en dehors de l'étendue la plus intérieure, on peut utiliser l'instruction nonlocal ; si elles ne sont pas déclarées nonlocal, ces variables sont en lecture seule (une tentative d'écriture dans une telle variable créera simplement une nouvelle variable locale dans l'étendue la plus intérieure, laissant la variable extérieure identique inchangée).

Habituellement, la portée locale fait référence aux noms locaux de la fonction courante (textuellement).

En dehors des fonctions, la portée locale fait référence au même espace de noms que la portée globale : l'espace de noms du module. Les définitions de classes placent encore un autre espace de noms dans la portée locale.

Il est important de réaliser que les portées sont déterminées textuellement : la portée globale d'une fonction définie dans un module est l'espace de noms de ce module, peu importe d'où ou par quel alias la fonction est appelée.

D'autre part, la recherche réelle des noms est faite dynamiquement, au moment de l'exécution - cependant, la définition du langage évolue vers une résolution statique des noms, au moment de la "compilation", donc ne comptez pas sur la résolution dynamique des noms ! (En fait, les variables locales sont déjà déterminées de manière statique).

**Une particularité de Python est que - si aucune déclaration globale ou non locale n'est en vigueur - les affectations aux noms vont toujours dans la portée la plus interne**.

Les affectations ne copient pas les données - elles lient simplement les noms aux objets. Il en va de même pour les suppressions : **l'instruction del x supprime la liaison de x de l'espace de nom** **référencé par la portée locale**. En fait, toutes les opérations qui introduisent de nouveaux noms utilisent la portée locale : en particulier, les instructions d'importation et les définitions de fonction lient le nom du module ou de la fonction dans la portée locale.

L'instruction globale peut être utilisée pour indiquer que des variables particulières vivent dans la portée globale et doivent y être rebasculées

l'instruction non locale indique que des variables particulières vivent dans une portée englobante et doivent y être rebasculées.

**9.2.1. Exemple de champs d'application et d'espaces de noms**

Cet exemple montre comment référencer les différents champs d'application et espaces de noms, et comment global et non local affectent la liaison des variables :

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Notez que l'affectation locale (qui est la valeur par défaut) n'a pas modifié la liaison de scope\_test avec le spam. L'affectation non locale a modifié l'affectation de spam de scope\_test, et l'affectation globale a modifié l'affectation au niveau du module.

Vous pouvez également constater qu'il n'y avait pas de liaison précédente pour le spam avant l'affectation globale.

**9.3. A First Look at Classes**

Les classes introduisent un peu de nouvelle syntaxe, trois nouveaux types d'objets et une nouvelle sémantique.

**9.3.1. Class Definition Syntax**

The simplest form of class definition looks like this:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Les définitions de classe, comme les définitions de fonction (instructions def) doivent être exécutées avant d'avoir un quelconque effet.

(On pourrait concevoir de placer une définition de classe dans une branche d'une instruction if, ou à l'intérieur d'une fonction).

**En pratique, les instructions à l'intérieur d'une définition de classe seront généralement des définitions de fonction, mais d'autres instructions sont autorisées, et parfois utiles - nous y reviendrons plus tard.**

**Les définitions de fonctions à l'intérieur d'une classe ont normalement une forme particulière de liste d'arguments, dictée par les conventions d'appel des méthodes** - encore une fois, ceci est expliqué plus tard.

**Lorsqu'une définition de classe est saisie, un nouvel espace de nom est créé, et utilisé comme portée locale** - ainsi, toutes les affectations aux variables locales vont dans ce nouvel espace de nom. En particulier, les définitions de fonction lient le nom de la nouvelle fonction ici.

Lorsqu'une définition de classe est quittée normalement (par la fin), un objet de classe est créé. Il s'agit essentiellement d'une enveloppe autour du contenu de l'espace de nom créé par la définition de classe ; nous en apprendrons davantage sur les objets de classe dans la section suivante. La portée locale originale (celle qui était en vigueur juste avant l'entrée de la définition de classe) est rétablie, et l'objet classe est lié ici au nom de classe donné dans l'en-tête de la définition de classe (ClassName dans l'exemple).