**4. More Control Flow Tools**

Outre l'instruction while que nous venons de présenter, Python utilise les instructions de contrôle de flux habituelles connues dans d'autres langages, avec quelques variantes.

4.1. if Statements

Perhaps the most well-known statement type is the [if](https://docs.python.org/3/reference/compound_stmts.html#if) statement. For example:

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

**Il peut y avoir zéro ou plusieurs parties elif, et la partie else est facultative.**

Le mot clé "elif" est l'abréviation de "else if" et est utile pour éviter une indentation excessive.

**Une séquence if ... elif ... elif ... est un substitut aux instructions switch ou case que l'on trouve dans d'autres langages.**

Si vous comparez la même **valeur à plusieurs constantes**, ou si vous vérifiez **des types ou des attributs spécifiques**, l'instruction **match** peut également vous être utile. Pour plus de détails, consultez les instructions match.

4.2. for Statements

L'instruction for en Python diffère quelque peu de celle à laquelle vous êtes peut-être habitué en C ou en Pascal.

Plutôt que de toujours itérer sur une progression arithmétique de nombres (comme en Pascal), ou de donner à l'utilisateur la possibilité de définir à la fois le pas d'itération et la condition d'arrêt (comme en C), **l'instruction for de Python itère sur les éléments d'une séquence (une liste ou une chaîne)**, **dans l'ordre où ils apparaissent dans la séquence**. Par exemple (sans jeu de mots) :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

**Le code qui modifie une collection tout en itérant sur cette même collection peut être délicat à mettre en œuvre. Il est généralement plus simple de boucler sur une copie de la collection ou de créer une nouvelle collection :**

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

4.3. The [range()](https://docs.python.org/3/library/stdtypes.html#range) Function

**Si vous devez itérer sur une séquence de nombres, la fonction intégrée range() est très utile. Elle génère des progressions arithmétiques :**

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

**Le point final donné ne fait jamais partie de la séquence générée ; range(10) génère 10 valeurs, les indices légaux pour les éléments d'une séquence de longueur 10.**

**Il est possible de laisser l'intervalle commencer à un autre nombre, ou de spécifier un incrément différent (même négatif ; on l'appelle parfois le 'pas') :**

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

Pour itérer sur les indices d'une séquence, vous pouvez combiner range() et len() comme suit :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

**Dans la plupart de ces cas, cependant, il est plus pratique d'utiliser la fonction enumerate(),** voir Techniques de bouclage.

Une chose étrange se produit si vous imprimez simplement une plage :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

À bien des égards, l'objet renvoyé par range() se comporte comme s'il s'agissait d'une liste, mais en fait, ce n'est pas le cas.

**C'est un objet qui renvoie les éléments successifs de la séquence souhaitée lorsque vous itérez sur lui, mais il ne constitue pas vraiment la liste, ce qui permet de gagner de la place.**

Nous disons qu'un tel objet est **itérable**, **c'est-à-dire qu'il convient comme cible pour les fonctions et les constructions qui attendent quelque chose à partir duquel elles peuvent obtenir des éléments** **successifs jusqu'à épuisement de la réserve.**

Nous avons vu que l'instruction for est une telle construction, tandis qu'un exemple de fonction qui prend un itérable est sum() :

Plus tard, nous verrons d'autres fonctions qui retournent des itérables et prennent des itérables comme arguments. Dans le chapitre Structures de données, nous discuterons plus en détail de list().

4.4. break and continue Statements, and else Clauses on Loops

**L'instruction break, comme en C, permet de sortir de la boucle for ou while la plus interne.**

Les instructions de boucle peuvent avoir une clause else ; elle est exécutée lorsque la boucle se termine par l'épuisement de l'itérable (avec for) ou lorsque la condition devient fausse (avec while), mais pas lorsque la boucle est terminée par une instruction **break**. Ceci est illustré par la boucle suivante, qui recherche des nombres premiers :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

**(Oui, c'est le bon code. Regardez bien : la clause else appartient à la boucle for, pas à l'instruction if).**

Lorsqu'elle est utilisée avec une boucle, la clause else a plus de points communs avec la clause else d'une instruction try qu'avec celle des instructions if : la clause else d'une instruction try s'exécute lorsqu'aucune exception ne se produit, et la clause else d'une boucle s'exécute lorsqu'aucune interruption ne se produit.

Pour plus d'informations sur l'instruction try et les exceptions, consultez la section Gestion des exceptions.

L'instruction **continue**, également empruntée au C, permet de passer à l'itération suivante de la boucle :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

4.5. pass Statements

L'instruction **pass** ne fait rien. Elle peut être utilisée lorsqu'une instruction est requise d'un point de vue syntaxique mais que le programme ne requiert aucune action. Par exemple :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

Cette méthode est couramment utilisée pour créer des classes minimales :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

Un autre endroit où **pass** peut être utilisé est en tant que substitut d'une fonction ou d'un corps conditionnel lorsque vous travaillez sur un nouveau code, ce qui vous permet de continuer à penser à un niveau plus abstrait. Le **pass** est ignoré en silence :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

4.6. match Statements

Une instruction **match** prend une expression et compare sa valeur à des motifs successifs donnés sous la forme d'un ou plusieurs blocs case.

Cela ressemble superficiellement à une instruction switch en C, Java ou JavaScript (et dans de nombreux autres langages), mais c'est plus proche du filtrage de motifs dans des langages comme Rust ou Haskell. Seul le premier motif qui correspond est exécuté et il peut également extraire des composants (éléments de séquence ou attributs d'objet) de la valeur dans des variables.

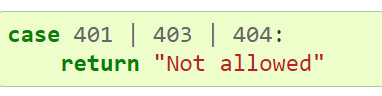
La forme la plus simple compare une valeur de sujet à un ou plusieurs littéraux :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

Notez le dernier bloc : le "nom de la variable" \_ agit comme un joker et ne manque jamais de correspondre. Si aucun cas ne correspond, aucune des branches n'est exécutée.

Vous pouvez combiner plusieurs littéraux dans un seul motif en utilisant | ("ou") :

****

Les modèles peuvent ressembler à des affectations de déballage, et peuvent être utilisés pour lier des variables :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

Vous pouvez utiliser des paramètres positionnels avec certaines classes intégrées qui fournissent un ordre pour leurs attributs (par exemple, les classes de données). Vous pouvez également définir une position spécifique pour les attributs dans les modèles en définissant l'attribut spécial \_\_match\_args\_\_ dans vos classes. S'il est défini à ("x", "y"), les modèles suivants sont tous équivalents (et lient tous l'attribut y à la variable var) :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Une façon recommandée de lire les motifs est de les considérer comme une forme étendue de ce que vous mettriez à gauche d'une affectation, afin de comprendre quelles variables seraient affectées à quoi. Seuls les noms autonomes (comme var ci-dessus) sont affectés par une instruction de correspondance. Les noms en pointillés (comme foo.bar), les noms d'attributs (les x= et y= ci-dessus) ou les noms de classes (reconnus par le "(...)" à côté d'eux comme Point ci-dessus) ne sont jamais assignés.

Les motifs peuvent être imbriqués de manière arbitraire. Par exemple, si nous avons une courte liste de points, nous pouvons la faire correspondre de la manière suivante :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

Nous pouvons ajouter une clause if à un motif, appelée "garde". Si la garde est fausse, match passe à l'essai du bloc de cas suivant. Notez que la capture de valeur a lieu avant que la garde ne soit évaluée :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

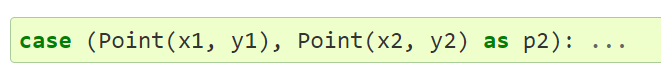
Plusieurs autres caractéristiques clés de cette déclaration :

Comme les affectations de déballage, les motifs tuple et list ont exactement la même signification et correspondent en fait à des séquences arbitraires. Une exception importante est qu'ils ne correspondent pas aux itérateurs ou aux chaînes de caractères.

Les motifs de séquence prennent en charge le déballage étendu : [x, y, \*rest] et (x, y, \*rest) fonctionnent de la même manière que les affectations de déballage. Le nom après \* peut également être \_, ainsi (x, y, \*\_) correspond à une séquence d'au moins deux éléments sans lier les éléments restants.

Modèles de mappage : {"bande passante" : b, "latence" : l} capture les valeurs "bande passante" et "latence" d'un dictionnaire. Contrairement aux motifs de séquence, les clés supplémentaires sont ignorées. Un déballage comme \*\*rest est également supporté. (Mais \*\*\_ serait redondant, il n'est donc pas autorisé).

Des sous-modèles peuvent être capturés en utilisant le mot-clé as :



capturera le deuxième élément de l'entrée sous la forme p2 (tant que l'entrée est une séquence de deux points).

La plupart des littéraux sont comparés par égalité, mais les singletons True, False et None sont comparés par identité.

Les modèles peuvent utiliser des constantes nommées. Ces noms doivent être en pointillés pour éviter qu'ils ne soient interprétés comme une variable de capture :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

Pour une explication plus détaillée et des exemples supplémentaires, vous pouvez consulter le document PEP 636 qui est rédigé sous forme de tutoriel.

4.7. Defining Functions

Nous pouvons créer une fonction qui écrit la série de Fibonacci sur une frontière arbitraire :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**Le mot-clé def introduit une définition de fonction. Il doit être suivi du nom de la fonction et de la liste des paramètres formels entre parenthèses.**

**Les instructions qui forment le corps de la fonction commencent à la ligne suivante et doivent être indentées.**

La première déclaration du corps de la fonction peut éventuellement être une chaîne de caractères littérale ; cette chaîne de caractères littérale est la chaîne de documentation de la fonction, ou docstring. (Il existe des outils qui utilisent les chaînes de documentation pour produire automatiquement de la documentation en ligne ou imprimée, ou pour permettre à l'utilisateur de naviguer de manière interactive dans le code ; c'est une bonne pratique d'inclure des chaînes de documentation dans le code que vous écrivez, alors prenez-en l'habitude.

L'exécution d'une fonction introduit une nouvelle table de symboles utilisée pour les variables locales de la fonction.

**Plus précisément, toutes les affectations de variables dans une fonction enregistrent la valeur dans la table des symboles locaux, tandis que les références de variables sont d'abord recherchées dans la table des symboles locaux, puis dans les tables des symboles locaux des fonctions englobantes, puis dans la table des symboles globaux, et enfin dans la table des noms intégrés.**

Ainsi, les variables globales et les variables de fonctions englobantes ne peuvent pas se voir attribuer directement une valeur au sein d'une fonction (sauf si, pour les variables globales, elles sont nommées dans une instruction globale ou, pour les variables de fonctions englobantes, dans une instruction non locale), mais elles peuvent être référencées.

Les paramètres réels (arguments) d'un appel de fonction sont introduits dans la table des symboles locaux de la fonction appelée lorsqu'elle est appelée ; ainsi, les arguments sont passés en utilisant l'appel par valeur (où la valeur est toujours une référence d'objet, et non la valeur de l'objet). 1 Lorsqu'une fonction appelle une autre fonction, ou s'appelle elle-même récursivement, une nouvelle table de symboles locale est créée pour cet appel.

Une définition de fonction associe le nom de la fonction à l'objet fonction dans la table de symboles courante. L'interpréteur reconnaît l'objet pointé par ce nom comme une fonction définie par l'utilisateur. D'autres noms peuvent également pointer vers ce même objet fonction et peuvent également être utilisés pour accéder à la fonction :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

Venant d'autres langages, vous pourriez objecter que fib n'est pas une fonction mais une procédure puisqu'elle ne renvoie pas de valeur.

En fait, même les fonctions sans instruction de retour renvoient une valeur, bien qu'elle soit plutôt ennuyeuse.

Cette valeur est appelée None (c'est un nom intégré). L'écriture de la valeur None est normalement supprimée par l'interpréteur si c'est la seule valeur écrite. Vous pouvez la voir si vous le voulez vraiment en utilisant print() :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

Il est simple d'écrire une fonction qui renvoie une liste des nombres de la série de Fibonacci, au lieu de l'imprimer :

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

Cet exemple, comme d'habitude, démontre certaines nouvelles fonctionnalités de Python :

L'instruction return retourne avec une valeur d'une fonction. return sans argument d'expression retourne None. Retourner à la fin d'une fonction renvoie également None.

**L'instruction result.append(a) appelle une méthode de l'objet liste result**.

Une méthode est une fonction qui "appartient" à un objet et qui est nommée obj.methodname, où obj est un objet quelconque (il peut s'agir d'une expression), et methodname est le nom d'une méthode définie par le type de l'objet.

Des types différents définissent des méthodes différentes. Les méthodes de différents types peuvent avoir le même nom sans causer d'ambiguïté.

(Il est possible de définir vos propres types d'objets et méthodes, en utilisant des classes, voir Classes) La méthode append() montrée dans l'exemple est définie pour les objets de type liste ; elle ajoute un nouvel élément à la fin de la liste. Dans cet exemple, elle est équivalente à result = result + [a], mais plus efficace.

4.8. More on Defining Functions

Il est également possible de définir des fonctions avec un nombre variable d'arguments.

**Il existe trois formes, qui peuvent être combinées.**

4.8.1. Default Argument Values

**La forme la plus utile** consiste à spécifier une valeur par défaut pour un ou plusieurs arguments. Cela crée une fonction qui peut être appelée avec moins d'arguments que ce qu'elle est censée permettre. Par exemple :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Cette fonction peut être appelée de plusieurs façons :

**en ne donnant que l'argument obligatoire : ask\_ok('Voulez-vous vraiment quitter ?')**

**en donnant l'un des arguments facultatifs : ask\_ok('OK to overwrite the file?', 2)**

**ou même en donnant tous les arguments : ask\_ok('OK to overwrite the file?', 2, 'Come on, only yes or no!')**

Cet exemple introduit également le mot-clé in. Celui-ci teste si une séquence contient ou non une certaine valeur.

Les valeurs par défaut sont évaluées au moment de la définition de la fonction dans la portée de la définition, de sorte que

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

will print 5.

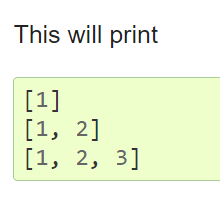
**Avertissement important :** La valeur par défaut n'est évaluée qu'une seule fois.

**Cela fait une différence lorsque la valeur par défaut est un objet mutable tel qu'une liste, un dictionnaire ou des instances de la plupart des classes.**

Par exemple, la fonction suivante accumule les arguments qui lui sont passés lors des appels suivants

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung



Si vous ne voulez pas que la valeur par défaut soit partagée entre les appels suivants, vous pouvez écrire la fonction comme ceci :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

4.8.2. Keyword Arguments

Les fonctions peuvent également être appelées à l'aide d'arguments de mots-clés de la forme **kwarg=valeur.** Par exemple, la fonction suivante :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

accepte un argument obligatoire (tension) et trois arguments facultatifs (état, action et type). Cette fonction peut être appelée de l'une des manières suivantes :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

mais tous les appels suivants seraient invalides :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**dans un appel de fonction, les arguments mots-clés doivent suivre les arguments positionnels.**

**Tous les arguments mots-clés passés doivent correspondre à l'un des arguments acceptés par la fonction (**par exemple, actor n'est pas un argument valide pour la fonction parrot), et leur ordre n'est pas important.

Cela inclut également les arguments non optionnels (par exemple, parrot(voltage=1000) est également valide).

Aucun argument ne peut recevoir une valeur plus d'une fois. Voici un exemple qui échoue à cause de cette restriction :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Lorsqu'un paramètre formel final de la forme **\*\*nom** est présent, il reçoit un dictionnaire (voir Mapping Types - dict) contenant tous les arguments des mots-clés sauf ceux correspondant à un paramètre formel.

Ceci peut être combiné avec un paramètre formel de la forme **\*name** (décrit dans la sous-section suivante) qui reçoit un tuple contenant les arguments positionnels au-delà de la liste des paramètres formels. **(\*nom** **doit apparaître avant \*\*nom.)** Par exemple, si nous définissons une fonction comme ceci :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**Notez que l'ordre dans lequel les arguments des mots-clés sont imprimés est garanti comme étant l'ordre dans lequel ils ont été fournis dans l'appel de la fonction.**

4.8.3. Special parameters

**Par défaut, les arguments peuvent être passés à une fonction Python soit par position, soit explicitement par mot-clé.**

Pour des raisons de lisibilité et de performance, il est judicieux de restreindre la manière dont les arguments peuvent être transmis de sorte qu'un développeur n'ait qu'à regarder la définition de la fonction pour déterminer si les éléments sont transmis par positionou par mot-clé.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

où / et \* sont facultatifs. S'ils sont utilisés, ces symboles indiquent le type de paramètre par lequel les arguments peuvent être transmis à la fonction : positionnel seulement, positionnel ou mot-clé, et mot-clé seulement. Les paramètres de mots-clés sont également appelés paramètres nommés.

4.8.3.1. Positional-or-Keyword Arguments

Si / et \* ne sont pas présents dans la définition de la fonction, les arguments peuvent être passés à une fonction par **position ou par mot-clé**.

4.8.3.2. Positional-Only Parameters

En regardant cela de plus près, il est possible de marquer certains paramètres comme étant uniquement positionnels. Si c'est le cas, l'ordre des paramètres est important et les paramètres ne peuvent pas être transmis par mot-clé.

Les paramètres positionnels sont placés avant un / (barre oblique). Le / est utilisé pour séparer logiquement les paramètres positionnels du reste des paramètres. S'il n'y a pas de / dans la définition de la fonction, il n'y a pas de paramètres positionnels.

Les paramètres qui suivent le / peuvent être positionnels ou mots-clés ou seulement.

4.8.3.3. Keyword-Only Arguments

Pour marquer les paramètres comme étant des mots-clés seulement, indiquant que les paramètres doivent être passés par un argument de mot-clé, placez un \* dans la liste des arguments juste avant le premier paramètre de mot-clé seulement.

4.8.3.4. Function Examples

Considérez les exemples suivants de définitions de fonctions en prêtant une attention particulière aux marqueurs / et \* :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

La première définition de fonction, standard\_arg, la forme la plus familière, n'impose aucune restriction sur la convention d'appel et les arguments peuvent être passés par position ou par mot-clé :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

La deuxième fonction pos\_only\_arg est restreinte pour n'utiliser que des paramètres positionnels car il y a un / dans la définition de la fonction :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Enfin, considérons cette définition de fonction qui présente une collision potentielle entre l'argument positionnel name et \*\*kwds qui a name comme clé :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Il n'y a pas d'appel possible qui renvoie True car le mot clé 'name' sera toujours lié au premier paramètre. Par exemple :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Mais en utilisant / (arguments positionnels uniquement), c'est possible puisqu'il autorise le nom comme argument positionnel et 'nom' comme clé dans les arguments de mots-clés :

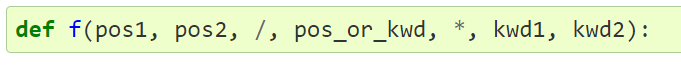
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

En d'autres termes, les noms des paramètres uniquement positionnels peuvent être utilisés dans \*\*kwds sans ambiguïté.

4.8.3.5. Recap

Le cas d'utilisation déterminera les paramètres à utiliser dans la définition de la fonction :



A titre indicatif :

Utilisez positional-only si vous voulez que le nom des paramètres ne soit pas disponible pour l'utilisateur. Ceci est utile lorsque les noms des paramètres n'ont pas de signification réelle, si vous voulez imposer l'ordre des arguments lorsque la fonction est appelée ou si vous devez prendre certains paramètres positionnels et des mots-clés arbitraires.

Utilisez les mots-clés uniquement lorsque les noms ont une signification et que la définition de la fonction est plus compréhensible si les noms sont explicites ou si vous voulez empêcher les utilisateurs de se fier à la position de l'argument transmis.

Pour une API, utilisez positional-only pour éviter de casser les changements d'API si le nom du paramètre est modifié dans le futur.

4.8.4. Arbitrary Argument Lists

Enfin, l'option la moins fréquemment utilisée consiste à spécifier qu'une fonction peut être appelée avec un nombre arbitraire d'arguments. Ces arguments seront regroupés dans un tuple (voir Tuples et séquences). Avant le nombre variable d'arguments, il peut y avoir zéro ou plusieurs arguments normaux.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Normalement, ces arguments variadiques seront les derniers dans la liste des paramètres formels, car ils regroupent tous les arguments d'entrée restants qui sont passés à la fonction. Tous les paramètres formels qui apparaissent après le paramètre \*args sont des arguments "keyword-only", ce qui signifie qu'ils ne peuvent être utilisés que comme mots-clés plutôt que comme arguments positionnels.

Ein Bild, das Text enthält.

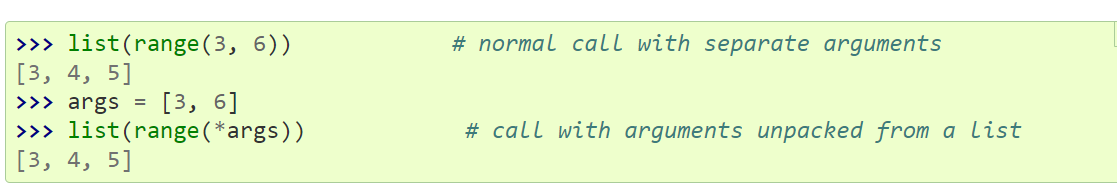
Automatisch generierte Beschreibung

4.8.5. Unpacking Argument Lists

La situation inverse se produit lorsque les arguments sont déjà dans une liste ou un tuple mais doivent être décomposés pour un appel de fonction nécessitant des arguments positionnels séparés.

Par exemple, la fonction intégrée range() attend des arguments de début et de fin séparés.

S'ils ne sont pas disponibles séparément, écrivez l'appel de fonction avec l'opérateur \* pour décompresser les arguments d'une liste ou d'un tuple :



Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

4.8.6. Lambda Expressions

De petites fonctions anonymes peuvent être créées à l'aide du mot-clé lambda. Cette fonction renvoie la somme de ses deux arguments : lambda a, b : a+b. Les fonctions lambda peuvent être utilisées partout où des objets fonctionnels sont nécessaires. Elles sont syntaxiquement limitées à une seule expression. Du point de vue sémantique, elles ne sont que du sucre syntaxique pour une définition de fonction normale. Comme les définitions de fonctions imbriquées, les fonctions lambda peuvent faire référence à des variables de la portée qui les contient :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

4.8.7. Documentation Strings

Voici quelques conventions concernant le contenu et le formatage des chaînes de documentation.

La première ligne doit toujours être un résumé court et concis de l'objectif de l'objet. Par souci de concision, elle ne doit pas indiquer explicitement le nom ou le type de l'objet, car ceux-ci sont disponibles par d'autres moyens (sauf si le nom est un verbe décrivant le fonctionnement d'une fonction). Cette ligne doit commencer par une majuscule et se terminer par un point.

S'il y a plusieurs lignes dans la chaîne de documentation, la deuxième ligne doit être vide, afin de séparer visuellement le résumé du reste de la description. Les lignes suivantes doivent être un ou plusieurs paragraphes décrivant les conventions d'appel de l'objet, ses effets secondaires, etc.

L'analyseur Python ne supprime pas l'indentation des chaînes de caractères de plusieurs lignes en Python, de sorte que les outils qui traitent la documentation doivent supprimer l'indentation si nécessaire. Ceci est fait en utilisant la convention suivante. La première ligne non vide après la première ligne de la chaîne détermine la quantité d'indentation pour l'ensemble de la chaîne de documentation. (Nous ne pouvons pas utiliser la première ligne, car elle est généralement adjacente aux guillemets ouvrants de la chaîne et son indentation n'est donc pas apparente dans le littéral de la chaîne). Les espaces blancs "équivalents" à cette indentation sont ensuite supprimés du début de toutes les lignes de la chaîne. Les lignes qui sont moins indentées ne doivent pas apparaître, mais si elles apparaissent, tous leurs espaces blancs de début doivent être supprimés. L'équivalence des espaces doit être testée après l'expansion des tabulations (jusqu'à 8 espaces, normalement).

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

4.8.8. Function Annotations

Les annotations de fonction sont des métadonnées totalement facultatives concernant les types utilisés par les fonctions définies par l'utilisateur (voir PEP 3107 et PEP 484 pour plus d'informations).

Les annotations sont stockées dans l'attribut \_\_annotations\_\_ de la fonction sous forme de dictionnaire et n'ont aucun effet sur les autres parties de la fonction. Les annotations de paramètres sont définies par un deux-points après le nom du paramètre, suivi d'une expression évaluant la valeur de l'annotation. Les annotations de retour sont définies par un -> littéral, suivi d'une expression, entre la liste des paramètres et les deux points indiquant la fin de l'instruction def. Dans l'exemple suivant, un argument obligatoire, un argument facultatif et la valeur de retour sont annotés :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

4.9. Intermezzo: Coding Style

Maintenant que vous êtes sur le point d'écrire des morceaux plus longs et plus complexes de Python, c'est le bon moment pour parler du style de codage. La plupart des langages peuvent être écrits (ou plus concis, formatés) dans différents styles ; certains sont plus lisibles que d'autres. Faciliter la lecture de votre code par les autres est toujours une bonne idée, et adopter un style de codage agréable y contribue grandement.

Pour Python, le PEP 8 s'est imposé comme le guide de style auquel la plupart des projets adhèrent ; il promeut un style de codage très lisible et agréable à l'œil. Chaque développeur Python devrait le lire à un moment donné ; voici les points les plus importants extraits pour vous :

* Utilisez une indentation de 4 espaces, et pas de tabulation.

Les 4 espaces sont un bon compromis entre une petite indentation (permet une plus grande profondeur d'imbrication) et une grande indentation (plus facile à lire). Les tabulations sont source de confusion et il est préférable de ne pas les utiliser.

* Faites en sorte que les lignes ne dépassent pas 79 caractères.

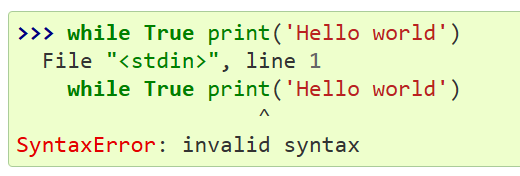
Cela aide les utilisateurs disposant de petits écrans et permet d'avoir plusieurs fichiers de code côte à côte sur des écrans plus grands.

* Utilisez des lignes vides pour séparer les fonctions et les classes, ainsi que les gros blocs de code à l'intérieur des fonctions.
* Dans la mesure du possible, placez les commentaires sur une ligne distincte.
* Utilisez les docstrings.
* Utilisez des espaces autour des opérateurs et après les virgules, mais pas directement à l'intérieur des constructions entre crochets : a = f(1, 2) + g(3, 4).
* Nommez vos classes et vos fonctions de manière cohérente ; la convention est d'utiliser la casse supérieure pour les classes et la casse inférieure avec les underscores pour les fonctions et les méthodes. Utilisez toujours self comme nom pour le premier argument de la méthode (voir Un premier regard sur les classes pour en savoir plus sur les classes et les méthodes).
* N'utilisez pas de codages fantaisistes si votre code est destiné à être utilisé dans des environnements internationaux. L'encodage par défaut de Python, UTF-8, ou même l'ASCII pur sont les meilleurs dans tous les cas.
* De même, n'utilisez pas de caractères non ASCII dans les identifiants s'il y a la moindre chance que des personnes parlant une autre langue lisent ou maintiennent le code.

# 8. Errors and Exceptions

Jusqu'à présent, les messages d'erreur n'ont été que mentionnés, mais si vous avez essayé les exemples, vous en avez probablement vu.

Il existe (au moins) deux types d'erreurs distinctes : **les erreurs de syntaxe et les exceptions.**



L'analyseur répète la ligne en question et affiche une petite "flèche" pointant vers le premier point de la ligne où l'erreur a été détectée. L'erreur est causée (ou du moins détectée) par le jeton qui précède la flèche : dans l'exemple, **l'erreur est détectée au niveau de la fonction print(),** car il manque les deux points (':') avant celle-ci. Le nom du fichier et le numéro de ligne sont imprimés pour que vous sachiez où regarder si l'entrée provient d'un script.

**8.2. Exceptions**

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**La dernière ligne du message d'erreur indique ce qui s'est passé.**

Les exceptions sont de différents types, et le type est imprimé dans le message : les types dans l'exemple sont **ZeroDivisionError**, **NameError** et **TypeError**.

**La chaîne imprimée comme type d'exception est le nom de l'exception intégrée qui s'est produite.** **Ceci est vrai pour toutes les exceptions intégrées**, mais ne doit pas nécessairement être vrai pour les **exceptions définies par l'utilisateur** (bien que ce soit une convention utile).

**Les noms d'exception standard sont des identifiants intégrés (et non des mots-clés réservés).**

Le reste de la ligne fournit des détails en fonction du type d'exception et de sa cause.

La partie précédente du message d'erreur montre le contexte dans lequel l'exception s'est produite, sous la forme d'un retour de pile.

En général, elle contient un retour de pile listant les lignes sources ; cependant, elle n'affichera pas les lignes lues depuis l'entrée standard.

Exceptions intégrées : liste des exceptions intégrées et leur signification.

JE DOIS METTRE LA LISTE DES EXCEPTIONS INTERGRE AET LEURS SIGNIFICATION A LA FIN DE CE CHAPITRE

**8.3. Handling Exceptions**

**Il est possible d'écrire des programmes qui gèrent certaines exceptions.**

Regardez l'exemple suivant, qui demande à l'utilisateur de saisir des données jusqu'à ce qu'un nombre entier valide ait été saisi, mais qui permet à l'utilisateur d'interrompre le programme (en utilisant Control-C ou tout autre moyen pris en charge par le système d'exploitation) ; **notez qu'une interruption générée par l'utilisateur est signalée en levant l'exception KeyboardInterrupt**.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**L'instruction try fonctionne comme suit.**

* Tout d'abord, la clause try (la ou les instructions situées entre les mots-clés try et except) est exécutée.
* Si aucune exception ne se produit, la clause except est ignorée et l'exécution de l'instruction try est terminée.
* Si une exception se produit pendant l'exécution de la clause try, le reste de la clause est ignoré. Ensuite, si son type correspond à l'exception nommée après le mot-clé except, la clause except est exécutée, puis l'exécution se poursuit après le bloc try/except.
* Si une exception qui ne correspond pas à l'exception nommée dans la clause except se produit, elle est transmise aux instructions try extérieures ; si aucun gestionnaire n'est trouvé, il s'agit d'une exception non gérée et l'exécution s'arrête avec un message comme indiqué ci-dessus.

Une instruction try peut comporter plus d'une clause except, afin de spécifier des gestionnaires pour différentes exceptions.

Un gestionnaire au maximum sera exécuté.

**Les gestionnaires ne gèrent que les exceptions qui apparaissent dans la clause except correspondante, et non dans d'autres gestionnaires de la même instruction try.**

Une clause except peut nommer plusieurs exceptions sous forme de tuple entre parenthèses, par exemple:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Une classe dans une clause except est compatible avec une exception si c'est la même classe ou une classe de base de celle-ci (mais pas l'inverse - une clause except listant une classe dérivée n'est pas compatible avec une classe de base). Par exemple, le code suivant imprimera B, C, D dans cet ordre :

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

**Notez que si les clauses d'exception étaient inversées (avec l'exception B en premier), l'impression aurait été B, B, B - la première clause d'exception correspondante est déclenchée.**

Lorsqu'une exception se produit, elle peut avoir des valeurs associées, également appelées arguments de l'exception.

**La présence et les types d'arguments dépendent du type d'exception.**

La clause except peut spécifier une variable après le nom de l'exception.

La variable est liée à l'instance d'exception qui possède généralement un attribut **args** qui stocke les arguments.

Par commodité, les types d'exception intégrés définissent **\_\_str\_\_()** pour imprimer tous les arguments sans accéder explicitement à **.args.**

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

La sortie \_\_str\_\_() de l'exception est imprimée comme la dernière partie ("détail") du message pour les exceptions non gérées.

**BaseException est la classe de base commune de toutes les exceptions.**

L'une de ses sous-classes, Exception, est la classe de base de toutes les exceptions non fatales.

Les exceptions qui ne sont pas des sous-classes de Exception ne sont généralement pas traitées, car elles sont utilisées pour indiquer que le programme doit se terminer.

Il s'agit notamment de **SystemExit**, qui est déclenchée par **sys.exit(),** et de **KeyboardInterrupt**, qui est déclenchée lorsqu'un utilisateur souhaite interrompre le programme.

Exception peut être utilisé comme un joker qui attrape (presque) tout.

Cependant, il est recommandé d'être aussi précis que possible quant aux types d'exceptions que l'on souhaite traiter et de permettre à toute exception inattendue de se propager.

**Le modèle le plus courant de traitement des exceptions consiste à imprimer ou à enregistrer l'exception, puis à la relancer (ce qui permet à l'appelant de traiter également l'exception**) :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

L'instruction try ... except comporte une clause else facultative qui, lorsqu'elle est présente, doit suivre toutes les clauses except. Elle est utile pour le code qui doit être exécuté si la clause try ne lève pas d'exception. Par exemple :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**L'utilisation de la clause else est préférable à l'ajout de code supplémentaire à la clause try, car elle évite d'attraper accidentellement une exception qui n'a pas été levée par le code protégé par l'instruction try ... except.**

Les gestionnaires d'exceptions ne traitent pas seulement les exceptions qui se produisent immédiatement dans la clause try, mais aussi celles qui se produisent à l'intérieur des fonctions qui sont appelées (même indirectement) dans la clause try. Par exemple

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**8.4. Raising Exceptions (Lever les exceptions)**

**L'instruction raise permet au programmeur de forcer une exception spécifiée à se produire.**

Par exemple :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Le seul argument de raise indique l'exception à lever.

Il doit s'agir soit d'une instance d'exception, soit d'une classe d'exception (une classe qui dérive de BaseException, comme Exception ou l'une de ses sous-classes).

Si une classe d'exception est passée, elle sera implicitement instanciée en appelant son constructeur sans argument :



Si vous devez déterminer si une exception a été levée mais que vous n'avez pas l'intention de la traiter, une forme plus simple de l'instruction raise vous permet de relancer l'exception :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**8.5. Exception Chaining**

Si une exception non traitée se produit à l'intérieur d'une section except, l'exception traitée lui sera attachée et incluse dans le message d'erreur :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Pour indiquer qu'une exception est la conséquence directe d'une autre, l'instruction **raise** permet d'ajouter une clause facultative **from** ::

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Cela peut être utile lorsque vous transformez des exceptions. Par exemple :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

t permet également de désactiver le chaînage automatique des exceptions en utilisant l'idiome from None :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

For more information about chaining mechanics, see [Built-in Exceptions](https://docs.python.org/3/library/exceptions.html#bltin-exceptions).

**8.6. User-defined Exceptions**

Les programmes peuvent nommer leurs propres exceptions en créant une nouvelle classe d'exception (voir Classes pour en savoir plus sur les classes Python).

Les exceptions doivent généralement être dérivées de la classe Exception, soit directement, soit indirectement.

Il est possible de définir des classes d'exception qui font tout ce que n'importe quelle autre classe peut faire, mais elles restent généralement simples, n'offrant souvent qu'un certain nombre d'attributs qui permettent aux gestionnaires de l'exception d'extraire des informations sur l'erreur.

**La plupart des exceptions sont définies avec des noms qui se terminent par "Error", de manière similaire à la dénomination des exceptions standard.**

De nombreux modules standard définissent leurs propres exceptions pour signaler les erreurs qui peuvent se produire dans les fonctions qu'ils définissent.

**8.7. Defining Clean-up Actions**

L'instruction try comporte une autre clause facultative destinée à définir les actions de nettoyage qui doivent être exécutées en toutes circonstances. Par exemple:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Si une clause finally est présente, elle s'exécutera en tant que dernière tâche avant la fin de l'instruction try.

La clause finally s'exécute que l'instruction try produise ou non une exception. Les points suivants traitent des cas plus complexes où une exception se produit :

* Si une exception se produit pendant l'exécution de la clause try, elle peut être traitée par une clause except. Si l'exception n'est pas traitée par une clause except, l'exception est soulevée à nouveau après l'exécution de la clause finally.
* Une exception peut se produire pendant l'exécution d'une clause except ou else. Là encore, l'exception est soulevée à nouveau après l'exécution de la clause finally.
* Si la clause finally exécute une instruction break, continue ou return, les exceptions ne sont pas levées à nouveau.
* Si l'instruction try atteint une instruction break, continue ou return, la clause finally sera exécutée juste avant l'exécution de l'instruction break, continue ou return.
* Si une clause finally comprend une instruction de retour, la valeur retournée sera celle de l'instruction de retour de la clause finally, et non celle de l'instruction de retour de la clause try.

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Comme vous pouvez le constater, la clause finally est exécutée dans tous les cas.

**L'erreur de type (TypeError) soulevée par la division de deux chaînes de caractères n'est pas traitée par la clause except et est donc soulevée à nouveau après l'exécution de la clause finally.**

**Dans les applications du monde réel, la clause finally est utile pour libérer des ressources externes (telles que des fichiers ou des connexions réseau), que l'utilisation de la ressource ait réussi ou non.**

**8.8. Predefined Clean-up Actions**

Certains objets définissent des actions de nettoyage standard à entreprendre lorsque l'objet n'est plus nécessaire, que l'opération utilisant l'objet ait réussi ou échoué. Regardez l'exemple suivant, qui tente d'ouvrir un fichier et d'imprimer son contenu à l'écran.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Le problème de ce code est qu'il laisse le fichier ouvert pendant un temps indéterminé après que cette partie du code a fini de s'exécuter.

Ce n'est pas un problème pour les scripts simples, mais il peut l'être pour les applications plus importantes.

L'instruction with permet d'utiliser des objets tels que les fichiers d'une manière qui garantit qu'ils sont toujours nettoyés rapidement et correctement.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Après l'exécution de l'instruction, le fichier f est toujours fermé, même si un problème a été rencontré lors du traitement des lignes. Les objets qui, comme les fichiers, fournissent des actions de nettoyage prédéfinies l'indiqueront dans leur documentation.

**8.9. Raising and Handling Multiple Unrelated Exceptions**

Il existe des situations où il est nécessaire de signaler plusieurs exceptions qui se sont produites.

C'est souvent le cas dans les cadres de concurrence, lorsque plusieurs tâches peuvent avoir échoué en parallèle, mais il existe également d'autres cas d'utilisation où il est souhaitable de poursuivre l'exécution et de collecter plusieurs erreurs plutôt que de lever la première exception.

Le groupe d'exceptions intégré (ExceptionGroup) regroupe une liste d'instances d'exceptions afin qu'elles puissent être levées ensemble. Il s'agit d'une exception en soi, qui peut donc être levée comme toute autre exception.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

En utilisant except\* au lieu de except, nous pouvons traiter de manière sélective uniquement les exceptions du groupe qui correspondent à un certain type. Dans l'exemple suivant, qui montre un groupe d'exceptions imbriquées, chaque clause except\* extrait du groupe les exceptions d'un certain type tout en laissant toutes les autres exceptions se propager vers d'autres clauses et éventuellement être relancées.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Notez que les exceptions imbriquées dans un groupe d'exceptions doivent être des instances, et non des types. En effet, dans la pratique, les exceptions sont généralement celles qui ont déjà été levées et rattrapées par le programme, selon le modèle suivant :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**8.10. Enriching Exceptions with Notes**

Lorsqu'une exception est créée afin d'être levée, elle est généralement initialisée avec des informations qui décrivent l'erreur qui s'est produite.

Dans certains cas, il est utile d'ajouter des informations après la levée de l'exception. À cette fin, les exceptions disposent d'une méthode add\_note(note) qui accepte une chaîne de caractères et l'ajoute à la liste des notes de l'exception.

Le rendu de la trace standard inclut toutes les notes, dans l'ordre où elles ont été ajoutées, après l'exception.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Par exemple, lorsque nous rassemblons des exceptions dans un groupe d'exceptions, nous pouvons souhaiter ajouter des informations contextuelles pour les erreurs individuelles. Dans ce qui suit, chaque exception du groupe est accompagnée d'une note indiquant quand cette erreur s'est produite

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**9. Classes**

**Portee = scope**

**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).

**9.3.2. Class Objects**

**Les objets de classe supportent deux types d'opérations : les références d'attributs et l'instanciation.**

Les références d'attributs utilisent la syntaxe standard utilisée pour toutes les références d'attributs en Python : obj.name.

Les noms d'attributs valides sont tous les noms qui se trouvaient dans l'espace de noms de la classe lorsque l'objet classe a été créé. Ainsi, si la définition de la classe ressemble à ceci :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

alors **MyClass.i et MyClass.f** sont des références d'attributs valides, renvoyant respectivement un entier et un objet fonction.

Les attributs de classe peuvent également être assignés, de sorte que vous pouvez modifier la valeur de MyClass.i par assignation. \_\_doc\_\_ est également un attribut valide, renvoyant la docstring appartenant à la classe : "Une classe d'exemple simple".

L'instanciation des classes utilise la notation des fonctions. Faites comme si l'objet de la classe était une fonction sans paramètre qui renvoie une nouvelle instance de la classe. Par exemple (en Supposita la classe ci-dessus) :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

crée une nouvelle instance de la classe et affecte cet objet à la variable locale x.

L'opération d'instanciation ("appel" d'un objet de classe) crée un objet vide.

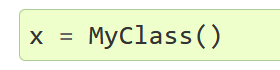
De nombreuses classes aiment créer des objets avec des instances personnalisées à un état initial spécifique.

**Par conséquent, une classe peut définir une méthode spéciale nommée \_\_init\_\_(),** comme ceci :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Lorsqu'une classe définit une méthode \_\_init\_\_(), l'instanciation de la classe invoque automatiquement \_\_init\_\_() pour l'instance de classe nouvellement créée. Ainsi, dans cet exemple, une nouvelle instance initialisée peut être obtenue par :



Bien sûr, la méthode \_\_init\_\_() peut avoir des arguments pour une plus grande flexibilité. Dans ce cas, les arguments donnés à l'opérateur d'instanciation de classe sont transmis à \_\_init\_\_(). Par exemple,

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**9.3.3. Instance Objects**

Maintenant, que pouvons-nous faire avec les objets d'instance ?

**Les seules opérations comprises par les objets d'instance sont les références d'attributs.**

**Il existe deux types de noms d'attributs valides : les attributs de données et les méthodes.**

Les attributs de données correspondent aux "variables d'instance" dans Smalltalk, et aux "membres de données" dans C++. Les attributs de données n'ont pas besoin d'être déclarés ; comme les variables locales, ils existent dès qu'ils sont affectés pour la première fois. Par exemple, si x est l'instance de MyClass créée ci-dessus, le morceau de code suivant imprimera la valeur 16, sans laisser de trace:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**L'autre type de référence d'attribut d'instance est une méthode.**

**Une méthode est une fonction qui "appartient" à un objet. (En Python, le terme méthode n'est pas propre aux instances de classes : d'autres types d'objets peuvent également avoir des méthodes.**

Par exemple, les objets de type liste possèdent des méthodes appelées append, insert, remove, sort, etc. Cependant, dans la discussion qui suit, nous utiliserons le terme méthode exclusivement pour désigner les méthodes des objets d'instance de classe, sauf mention contraire explicite).

Les noms de méthodes valides d'un objet d'instance dépendent de sa classe.

Par définition, tous les attributs d'une classe qui sont des objets fonctionnels définissent les méthodes correspondantes de ses instances.

Ainsi, dans notre exemple, x.f est une référence de méthode valide, puisque MyClass.f est une fonction, mais x.i ne l'est pas, puisque MyClass.i ne l'est pas. Mais x.f n'est pas la même chose que MyClass.f - **c'est un objet méthode, pas un objet fonction**.

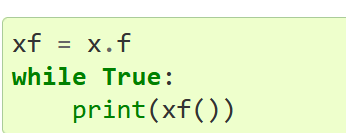
**9.3.4. Method Objects**

En général, une méthode est appelée juste après avoir été liée

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Dans l'exemple de MyClass, cela renverra la chaîne 'hello world'. Cependant, il n'est pas nécessaire d'appeler une méthode immédiatement : x.f est un objet méthode, et peut être stocké et appelé plus tard. Par exemple :



continuera à imprimer hello world jusqu'à la fin des temps.

Que se passe-t-il exactement lorsqu'une méthode est appelée ? Vous avez peut-être remarqué que x.f() a été appelée sans argument ci-dessus, même si la définition de la fonction f() spécifie un argument.

Que s'est-il passé avec l'argument ? **Python lève certainement une exception lorsqu'une fonction qui requiert un argument est appelée sans en avoir - même si l'argument n'est pas réellement utilisé**.

**En fait, vous avez peut-être deviné la réponse : la particularité des méthodes est que l'objet d'instance est passé comme premier argument de la fonction.**

**Dans notre exemple, l'appel x.f() est exactement équivalent à MyClass.f(x).**

**En général, l'appel d'une méthode avec une liste de n arguments est équivalent à l'appel de la fonction correspondante avec une liste d'arguments qui est créée en insérant l'objet d'instance de la méthode avant le premier argument.**

Si vous ne comprenez toujours pas le fonctionnement des méthodes, un regard sur l'implémentation peut peut-être clarifier les choses.

Lorsqu'un attribut non-donné d'une instance est référencé, la classe de l'instance est recherchée.

Si le nom désigne un attribut de classe valide qui est un objet fonction, un objet méthode est créé en regroupant (des pointeurs vers) l'objet instance et l'objet fonction qui vient d'être trouvé dans un objet abstrait : c'est l'objet méthode.

Lorsque l'objet méthode est appelé avec une liste d'arguments, une nouvelle liste d'arguments est construite à partir de l'objet instance et de la liste d'arguments, et l'objet fonction est appelé avec cutte nouvelle liste d'arguments.

**9.3.5. Class and Instance Variables**

D'une manière générale, les variables d'instance sont destinées aux données propres à chaque instance et les variables de classe sont destinées aux attributs et méthodes partagés par toutes les instances de la classe :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Comme nous l'avons vu dans Un mot sur les noms et les objets, les données partagées peuvent avoir des effets surprenants lorsqu'elles impliquent des objets mutables tels que des listes et des dictionnaires.

**Par exemple, la liste d'astuces dans le code suivant ne devrait pas être utilisée comme variable de classe car une seule liste serait partagée par toutes les instances de Dog :**

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

Une conception correcte de la classe devrait utiliser une variable d'instance à la place :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**9.4. Random Remarks**

**Si le même nom d'attribut apparaît à la fois dans une instance et dans une classe, la** recherche d'attribut donne la priorité à l'instance :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Les attributs de données peuvent être référencés par les méthodes ainsi que par les utilisateurs ordinaires ("clients") d'un objet.

En d'autres termes, les classes ne sont pas utilisables pour implémenter des types de données abstraits purs.

En fait, rien dans Python ne permet d'imposer le masquage des données - tout est basé sur la convention. (D'un autre côté, l'implémentation de Python, écrite en C, peut complètement cacher les détails de l'implémentation et contrôler l'accès à un objet si nécessaire ; ceci peut être utilisé par les extensions de Python écrites en C).

Les clients doivent utiliser les attributs de données avec précaution - les clients peuvent perturber les invariants maintenus par les méthodes en estampillant leurs attributs de données.

Notez que les clients peuvent ajouter leurs propres attributs de données à un objet d'instance sans affecter la validité des méthodes, tant que les conflits de noms sont évités - encore une fois, une convention de nommage peut éviter beaucoup de maux de tête ici.

Il n'existe pas de raccourci pour référencer les attributs de données (ou d'autres méthodes !) à l'intérieur des méthodes.

Je trouve que cela augmente la lisibilité des méthodes : il **n'y a aucune chance de confondre les variables locales et les variables d'instance en parcourant une méthode**.

**Souvent, le premier argument d'une méthode est appelé self. Ce n'est rien de plus qu'une convention : le nom self n'a absolument aucune signification particulière pour Python**.

Notez cependant qu'en ne respectant pas cette convention, votre code peut être moins lisible pour les autres programmeurs Python, et il est également concevable qu'un programme de navigateur de classes puisse être écrit en s'appuyant sur une telle convention.

Tout objet fonction qui est un attribut de classe définit une méthode pour les instances de cette classe.

Il n'est pas nécessaire que la définition de la fonction soit textuellement incluse dans la définition de la classe : l'affectation d'un objet fonction à une variable locale de la classe est également acceptable. Par exemple:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Maintenant, f, g et h sont tous des attributs de la classe C qui font référence à des objets fonctions, et par conséquent,

ils sont tous des méthodes d'instances de C - h étant exactement équivalent à g.

**Notez que cette pratique ne sert généralement qu'à embrouiller le lecteur d'un programme.**

Les méthodes peuvent appeler d'autres méthodes en utilisant les attributs de méthode de l'argument self :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**Les méthodes peuvent faire référence à des noms globaux de la même manière que les fonctions ordinaires.**

La portée globale associée à une méthode est le module contenant sa définition. (Une classe n'est jamais utilisée comme portée globale.)

Bien que l'on rencontre rarement une bonne raison d'utiliser des données globales dans une méthode, il existe de nombreuses utilisations légitimes de la portée globale : d'une part, les fonctions et modules importés dans la portée globale peuvent être utilisés par les méthodes, ainsi que les fonctions et classes qui y sont définies.

Habituellement, la classe contenant la méthode est elle-même définie dans cette portée globale, et dans la section suivante, nous trouverons quelques bonnes raisons pour lesquelles une méthode voudrait référencer sa propre classe.

Chaque valeur est un objet, et possède donc une classe (également appelée type). Elle est stockée sous la forme object.\_\_class\_\_.

**9.5. Inheritance**

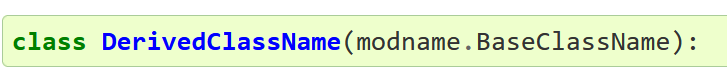
Bien entendu, une fonctionnalité du langage ne serait pas digne du nom de "classe" sans supporter l'héritage. La syntaxe de la définition d'une classe dérivée ressemble à ceci :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Le nom BaseClassName doit être défini dans une portée contenant la définition de la classe dérivée. Au lieu d'un nom de classe de base, d'autres expressions arbitraires sont également autorisées. Cela peut être utile, par exemple, lorsque la classe de base est définie dans un autre module :

**Prtee = Scope juste pour Info**



L'exécution de la définition d'une classe dérivée se déroule de la même manière que pour une classe de base.

Lorsque l'objet classe est construit, la classe de base est mémorisée.

Elle est utilisée pour résoudre les références aux attributs : si un attribut demandé n'est pas trouvé dans la classe, la recherche se poursuit dans la classe de base.

**Cette règle est appliquée de manière récursive si la classe de base est elle-même dérivée d'une autre classe.**

L'instanciation des classes dérivées n'a rien de particulier : DerivedClassName() crée une nouvelle instance de la classe.

**Les références aux méthodes sont résolues de la manière suivante : l'attribut de classe correspondant est recherché, en descendant la chaîne des classes de base si nécessaire, et la référence à la méthode est valide si elle donne un objet fonction**.

Les classes dérivées peuvent remplacer les méthodes de leurs classes de base.

Comme les méthodes n'ont pas de privilèges particuliers lorsqu'elles appellent d'autres méthodes du même objet, une méthode d'une classe de base qui appelle une autre méthode définie dans la même classe de base peut finir par appeler une méthode d'une classe dérivée qui la surcharge.

(Pour les programmeurs C++ : toutes les méthodes en Python sont effectivement virtuelles).

Une méthode prioritaire dans une classe dérivée peut en fait vouloir étendre plutôt que de simplement remplacer la méthode de la classe de base du même nom.

Il existe un moyen simple d'appeler directement la méthode de la classe de base : il suffit d'appeler **BaseClassName.methodname(self, arguments).** Ceci est parfois utile pour les clients également. (Notez que cela ne fonctionne que si la classe de base est accessible en tant que BaseClassName dans la portée globale).

**Python possède deux fonctions intégrées qui fonctionnent avec l'héritage :**

Utilisez **isinstance()** pour vérifier le type d'une instance : **isinstance(obj, int)** sera True seulement si **obj.\_\_class\_\_** est int ou une classe dérivée de int.

Utilisez **issubclass()** pour vérifier l'héritage des classes : **issubclass(bool, int**) est True puisque bool est une sous-classe de int. (je crois qu il disent cela car 0 et 1 sont des entiers)

Cependant, **issubclass(float, int**) est False puisque float n'est pas une sous-classe de int.

**9.5.1. Multiple Inheritance**

Python supporte également une forme d'héritage multiple. Une définition de classe avec plusieurs classes de base ressemble à ceci :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Dans la plupart des cas, et dans les cas les plus simples, vous pouvez considérer la recherche d'attributs hérités d'une classe parent comme une recherche en profondeur d'abord, de gauche à droite, sans chercher deux fois dans la même classe lorsqu'il y a un chevauchement dans la hiérarchie.

**Ainsi, si un attribut n'est pas trouvé dans DerivedClassName, il est recherché dans Base1, puis (récursivement) dans les classes de base de Base1, et s'il n'y a pas été trouvé, il est recherché dans Base2, et ainsi de suite.**

En fait, c'est un peu plus complexe que cela ; l'ordre de résolution des méthodes change dynamiquement pour supporter les appels coopératifs à **super()**.

Cette approche est connue dans certains autres langages à héritage multiple sous le nom de call-next-method et est plus puissante que l'appel à super() que l'on trouve dans les langages à héritage unique.

L'ordonnancement dynamique est nécessaire car tous les cas d'héritage multiple présentent une ou plusieurs relations en diamant (où au moins une des classes parentes est accessible par plusieurs chemins depuis la classe la plus basse).

Par exemple, toutes les classes héritent d'object, donc tout cas d'héritage multiple fournit plus d'un chemin pour atteindre object.

Pour éviter d'accéder aux classes de base plus d'une fois, l'algorithme dynamique linéarise l'ordre de recherche de manière à préserver l'ordre de gauche à droite spécifié dans chaque classe, à n'appeler chaque parent qu'une seule fois et à être monotone (ce qui signifie qu'une classe peut être sous-classée sans affecter l'ordre de préséance de ses parents).

Ensemble, ces propriétés permettent de concevoir des classes fiables et extensibles avec un héritage multiple. Pour plus de détails, voir https://www.python.org/download/releases/2.3/mro/.

**9.6. Private Variables**

**Les variables d'instance "privées", auxquelles on ne peut accéder que depuis l'intérieur d'un objet, n'existent pas en Python.**

Cependant, il existe une convention qui est suivie par la plupart du code Python: un nom préfixé par un trait de soulignement **(par exemple \_spam**) **doit être traité comme une partie non publique de l'API (qu'il s'agisse d'une fonction, d'une méthode ou d'un membre de données).**

**Elle doit être considérée comme un détail de mise en œuvre et peut être modifiée sans préavis.**

Puisqu'il existe un cas d'utilisation valide pour les membres privés de classe (à savoir éviter les conflits de noms avec les noms définis par les sous-classes),

il existe un support limité pour un tel mécanisme, appelé name mangling.

Tout identifiant de la forme **\_\_spam** (au moins deux traits de soulignement en tête, au plus un trait de soulignement en queue) est remplacé textuellement **par \_classname\_\_**spam, où classname est le nom de la classe courante, sans le(s) trait(s) de soulignement en tête.

Ce traitement est effectué sans tenir compte de la position syntaxique de l'identificateur, tant qu'il se trouve dans la définition d'une classe.

**La gestion des noms est utile pour permettre aux sous-classes de remplacer des méthodes sans rompre les appels de méthode intra-classe**. Par exemple:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

L'exemple ci-dessus fonctionnerait même si MappingSubclass devait introduire un identifiant \_\_update puisqu'il est remplacé par \_Mapping\_\_update dans la classe Mapping et \_MappingSubclass\_\_update dans la classe MappingSubclass respectivement.

Notez que les règles d'altération sont principalement conçues pour éviter les accidents ;

il est toujours possible d'accéder ou de modifier une variable considérée comme privée. Cela peut même être utile dans des circonstances particulières, comme dans le débogueur.

**Notez que le code passé à exec() ou eval() ne considère pas le nom de classe de la classe invoquante comme étant la classe courante ; ceci est similaire à l'effet de l'instruction global, dont l'effet est également limité au code qui est compilé par octet. La même restriction s'applique à getattr(), setattr() et delattr(), ainsi qu'à la référence directe à \_\_dict\_\_.directement.**

**9.7. Odds and Ends**[**¶**](https://docs.python.org/3/tutorial/classes.html#odds-and-ends)

Il est parfois utile de disposer d'un type de données similaire au "record" du Pascal ou au "struct" du C, regroupant quelques éléments de données nommés.

L'approche idiomatique consiste à utiliser des classes de données à cette fin :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Un morceau de code Python qui s'attend à un type de données abstrait particulier peut souvent se voir passer une classe qui émule les méthodes de ce type de données à la place.

Par exemple, si vous avez une fonction qui formate des données à partir d'un objet fichier, vous pouvez définir une classe avec les méthodes read() et readline() qui récupèrent les données à partir d'un tampon de type chaîne de caractères, et les passer comme argument.

Les objets méthodes d'instance ont également des attributs : m.\_\_self\_\_ est l'objet d'instance avec la méthode m(), et m.\_\_func\_\_ est l'objet fonction correspondant à la méthode.

**9.8. Iterators**

Vous avez probablement remarqué que la plupart des objets conteneurs peuvent être parcourus en boucle à l'aide d'une instruction for :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ce style d'accès est clair, concis et pratique. L'utilisation d'itérateurs est omniprésente et unifie Python.

En coulisse, l'instruction for appelle iter() sur l'objet conteneur.

La fonction renvoie un objet itérateur qui définit la méthode \_\_next\_\_() qui accède aux éléments du conteneur un par un.

Lorsqu'il n'y a plus d'éléments, \_\_next\_\_() lève une exception StopIteration qui indique à la boucle for de se terminer. Vous pouvez appeler la méthode \_\_next\_\_() à l'aide de la fonction intégrée next() ; cet exemple montre comment tout cela fonctionne :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Après avoir vu les mécanismes du protocole des itérateurs, il est facile d'ajouter le comportement des itérateurs à vos classes. Définissez une méthode \_\_iter\_\_() qui renvoie un objet avec une méthode \_\_next\_\_(). Si la classe définit \_\_next\_\_(), alors \_\_iter\_\_() peut simplement retourner self :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**9.9. Generators**

**Les générateurs sont un outil simple et puissant pour créer des itérateurs.**

Ils s'écrivent comme des fonctions ordinaires, mais utilisent l'instruction **yield** chaque fois qu'ils veulent renvoyer des données.

Chaque fois que la fonction **next()** est appelée, le générateur reprend là où il s'est arrêté (il se souvient de toutes les valeurs des données et de la dernière instruction exécutée).

Un exemple montre que les générateurs peuvent être trivialement faciles à créer :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Tout ce qui peut être fait avec des générateurs peut également être fait avec des itérateurs basés sur des classes, comme décrit dans la section précédente.

Ce qui rend les générateurs si compacts est que les méthodes \_\_iter\_\_() et \_\_next\_\_() sont créées automatiquement.

**Une autre caractéristique clé est que les variables locales et l'état d'exécution sont automatiquement sauvegardés entre les appels.**

Cela a rendu la fonction plus facile à écrire et beaucoup plus claire qu'une approche utilisant des variables d'instance comme self.index et self.data.

En plus de la création automatique de méthodes et de la sauvegarde de l'état du programme, lorsque les générateurs se terminent, ils lèvent automatiquement **StopIteration.**

**La combinaison de ces fonctionnalités permet de créer facilement des itérateurs sans effort supplémentaire par rapport à l'écriture d'une fonction ordinaire.**

**9.10. Generator Expressions**

Certains générateurs simples peuvent être codés succinctement comme des expressions utilisant une syntaxe similaire à celle des compréhensions de listes, mais avec des parenthèses au lieu de crochets.

Ces expressions sont conçues pour les situations où le générateur est utilisé immédiatement par une fonction englobante.

Les expressions de générateur sont plus compactes mais moins polyvalentes que les définitions de générateur complètes et ont tendance à être plus respectueuses de la mémoire que les compréhensions de liste équivalentes.

Exemples :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung