Initiation à Python - leçon 3.1.3

s1
Dans cette séquence, nous allons découvrir les types composés. Les types composés permettent de manipuler des collections d'éléments de type de base.

---
s2

Dans la leçon précédente nous avons étudié les types de base : booléen, entiers, flottants, complexes, chaînes de caractères. Les types composés sont des conteneurs de ces types de base ; ils permettent de créer et de manipuler des collections d'éléments, comme par exemple ici un tableau de dates ou une liste d'auteurs.

Du point de vue informatique, les types composés ne contiennent pas à proprement parler les objets, mais des références vers les objets en questions, comme dans l'exemple présenté ici : la variable a référence un objet de type composé contenant des références pointant vers les trois objets de type int 2011, 2012, 2013 etc.
.

---
s3

Nous allons faire la distinction entre les collections ordonnées et les collections non ordonnées.

Dans la première partie de cette leçon, nous allons introduire les collections ordonnées. Dans ces collections, comme leur nom l'indique, l'ordre des éléments est important. Nous allons successivement introduire les listes, les tuples (qui sont des collections non modifiables une fois définies) et nous allons reparler des chaînes de caractères.

Dans une seconde partie, nous introduirons les listes non ordonnées : les objets de type set (qui sont des collections dans lesquelles on ne peut pas trouver deux fois le même élément) et la table de hachage (objet de type dictionnaire) qui permettent d'associer une clé et une valeur à la manière par exemple d'un annuaire téléphonique qui associe un nom et un numéro.

Chacune de ces collections a son intérêt. Il est donc important de bien connaître ces types composés pour faire le choix de la collection la plus pertinente pour un problème donné.

Nous verrons également plus loin que toutes ces collections possèdent une propriété intéressante : on peut les parcourir afin d'accéder à l'ensemble des éléments.

---
s4

Commençons par les listes. Une liste est une collection ordonnée d'éléments. C'est un simple tableau. Les listes sont des objets mutables, c'est-à-dire que leur contenu peut être modifié. Nous allons voir ça.

La syntaxe pour déclarer une liste est la suivante : des éléments entre crochets. Bien entendu, nous pouvons créer des listes de listes. Nous verrons ça dans les exercices.

Voici les opérations possibles sur les listes ainsi que quelques exemples.

Les exercices proposés dans le document texte associé à cette vidéo vous aideront à vous familiariser avec ces opérateurs et ces fonctions.

La première ligne de ce tableau représente la syntaxe permettant de déclarer une liste contenant des éléments.

La deuxième ligne représente la syntaxe permettant de déclarer une liste vide.

Voici l'opération de concaténation entre listes. Attention, une liste n'est pas un vecteur, comme le montre l'exemple ci-dessous.

Voici la syntaxe qui permet d'accéder à une valeur de la liste.

Les deux lignes suivantes permettent de construire des sous-listes à partir d'une liste initiale, comme dans cet exemple où l'on crée une liste constituée des 3 premiers éléments de a.

L'instruction suivante permet de savoir si un élément appartient à une liste.

all (respectivement any) est une fonction qui renvoie True si tous les éléments (respectivement chacun des éléments) d'une liste est True. Si la liste est vide ces deux fonctions renvoient False.

len permet de connaître le nombre d'éléments de la liste.

Enfin, str permet de transformer une liste en chaîne de caractères en concaténant tous ses éléments.

Rappelons enfin qu'une liste est un objet mutable (c'est-à-dire qui peut être modifié). Dans cet exemple, on change la première valeur de a.

Un mot sur la fonction intrinsèque range qui permet de créer une liste d'entiers. Cette fonction admet plusieurs paramètres : on peut simplement indiquer un nombre de valeurs, on peut indiquer le début et la fin ; on peut indiquer le début, la fin et le pas, comme dans les exemples présentés.

Je vous propose de reproduire les exemples présentés.

---

s5

Une chaîne de caractères est une liste dans laquelle chaque caractère de la chaîne est associé à un entier, en commençant par zéro. On peut même étendre ce tableau vers la gauche avec des index négatifs.
Ainsi a[3] représente le 4e caractère de la chaîne a, c'est-à-dire r ; a[-5] représente le caractère avec l'index -5, c'est-à-dire A.
L'expression suivante permet d'extraire la sous-chaîne comprise entre les index 1 inclus et 4 exclus.
Je vous propose de reproduire les exemples présentés et vous laisse tester la fonction len() qui renvoie le nombre d'éléments d'un tableau.

---

s6

Un tuple est une collection un peu particulière, puisqu'elle est non mutable (non modifiable). Voyons cela sur un exemple. La syntaxe pour déclarer un tuple est similaire à celle de la liste, sauf que dans le cas d'un tuple on a des parenthèses.
Non mutable, signifie qu'une fois définie ses éléments ne supportent pas la réaffectation. En revanche, ses éléments mutables peuvent être modifiés, comme dans cet exemple où le troisième élément est une liste.

Intérêt : l'assurance que ne sera pas modifiée par erreur dans le programme (par exemple tableau des années entre 2000 et 2050)

Voici les opérations possibles sur les tuples. Ce sont pratiquement les mêmes que pour les listes.

Je vous laisse reproduire les exemples présentés.

---

s7

La chaîne de caractères est certes un type de base, mais peut être vue également comme une collection ordonnée et non mutable de caractères. On peut bien entendu ajouter un caractère (ou une autre chaîne) par concaténation à une chaîne existante, mais le résultat est la création d'une nouvelle chaîne à une nouvelle adresse en mémoire.

---

s8

Examinons à présent les sets. Un set est une collection dans laquelle les éléments sont uniques ; de plus cette collection n'est pas ordonnée.
Voyons cela sur un exemple. La syntaxe pour déclarer un set est similaire à celle de la liste, sauf que dans le cas d'un set on a des accolades ; dans cette façon de déclarer, il faut évidemment que tous les éléments soient disctincts, sinon l'interpréteur affichera une erreur.
Une autre façon de procéder est d'utiliser la fonction set() avec en argument un objet de type liste. En d'autres termes, pour créer un set, on peut commencer par créer une liste que l'on transforme en set.
Dans l'exemple présenté, on a créé une liste contenant deux fois l'élément 1. Au passage, on aurait pu passer directement la liste [1, 2, 3, 1] en paramètre de la fonction set. L'objet set ainsi créé ne comporte plus l'élément 1 qu'une fois.
Que veut dire non ordonnée ? Les listes et les tuples sont des collections ordonnées dans le sens où les éléments sont à une place bien particulière définie par leur indice et c'est d'ailleurs par leur indice qu'on peut les retrouver.
Dans un set, nous n'avons pas cette possibilité.
L'intérêt des sets réside dans certaines opérations. Voici ces opérations possibles sur les sets. On voit qu'il est également possible de créer un set à partir d'un objet chaîne de caractères et d'un objet table - que nous découvrirons dans la planche suivante.

Voici un exemple d'intersection de deux sets.
Voici un exemple d'union de deux sets.
Voici un exemple de soustraction entre deux sets. Le résultat étant un set constitué d'éléments présents dans la première liste, mais pas dans la seconde.
Voici un exemple de différence symétrique entre deux sets. Le résultat étant un set constitué d'éléments présents dans la première liste ou dans la seconde, mais pas dans les deux.

Voici un exemple d'utilisation de la méthode issubset() qui permet de savoir si un set est un sous-ensemble d'un autre set.
Le fait que les sets soient non ordonnés est une conséquence de l'existence de ces opérations.

Enfin, on retrouve les opérateurs in et not in dont le résultat est un booléen, ainsi que l'opérateur len qui fournit la taille du set et l'opérateur str qui convertit un set en une chaîne de caractères.

Je vous propose de reproduire les exemples présentés.

---
s9

Examinons les tables. Une table (autrement appelée dictionnaire) définit une collection non ordonnée d'éléments ou plutôt de couples au sein desquels, l'un des deux - appelé clé - va permettre de retrouver le second - appelé valeur.
Pour comprendre cette logique, il faut penser à un annuaire téléphonique, dans lequel les entrées - les clés - sont les noms de personnes, et la valeur leur numéro.

Dans cet exemple, les clés sont les noms, la valeur est également une chaîne de caractères, mais les clés peuvent être de tout type de base et les valeurs de tout type.

Voici la syntaxe pour retrouver un des éléments de la table.

Voici la liste des opérations possibles sur les tables. Pas de nouveauté. On remarquera seulement qu'il n'existe aucune opération de concaténation de tables.

Signalons deux méthodes importantes d'un objet de type dic. La méthode keys() qui permet de créer la liste de toutes les clés. La méthode values() qui permet de créer la liste de toutes les valeurs.

Je vous propose de reproduire les exemples présentés.

---
s10

Voici donc pour une présentation rapide des types composés : nous avons introduit les collections ordonnées (listes et tuples) et les collections non ordonnées (sets et tables de hachage).
La prochaine leçon sera l’occasion d’entrer dans la syntaxe du langage Python.