- Séquences : *list*, *tuple*, *str*
- Ensembles : *set*, *frozenset*
- Dictionnaire : *dict*

Séquences

La *list*, le *tuple* et la *str* sont des séquences en Python. Dans une séquence, l'ordre est important !

La *list* est la plus générale (et la plus utilisée). Elle représente une séquence d'objets arbitraires.

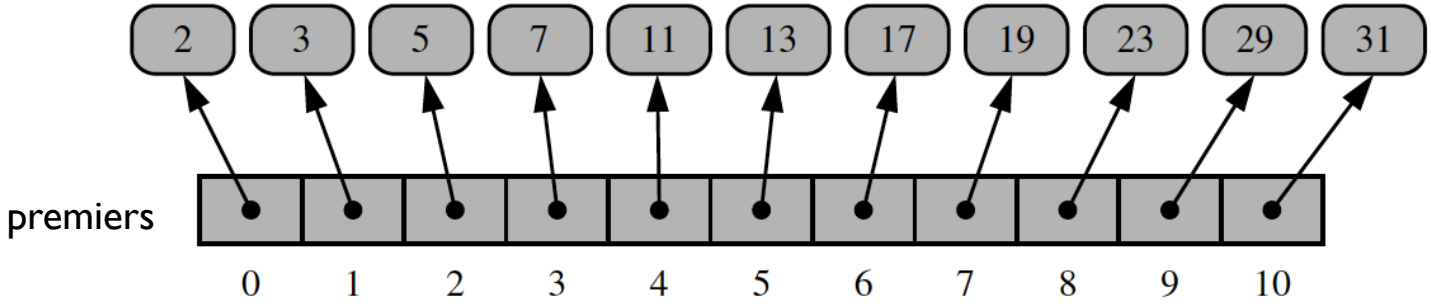
Le *tuple* est la version immuable de la *list* qui peut donc bénéficier d'une représentation interne simplifiée.

La *str* a été spécialement conçue pour représenter une séquence immuable de caractères. En Python, il n'y a pas de type indépendant pour les caractères. Ils sont des *str* de longueur 1.

list

Une *list* est pour stocker une séquence d'objets, soit, techniquement, une séquence de références à ses éléments.

Par exemple, la représentation interne en Python pour une liste d'entiers, instanciée par
premiers = [2,3,5,7,11,13,17,19,23,29,31], est la suivante :



où les indices des éléments sont indiqués en dessous de chaque entrée.

La *list* est générique, ses éléments sont des objets arbitraires.

La *list* est une séquence basée sur un tableaux indexés à partir de zéro, donc une *list* de longueur *n* possède des éléments indexés de *0* à *n-1* inclus, ou *0* à *n* exclu.

La *list* est la structure de données intégrée la plus utilisée en Python. Elle sera centrale pour notre étude des structures de données.

La *list* possède la capacité d'étendre et de contracter sa capacité dynamiquement et selon les besoins.

Python utilise les caractères [] comme délimiteurs pour un littéral de *list*, où [] représente une *list* vide.

Comme autre exemple, [‘rouge’, ‘vert’, ‘bleu’] est une *list* contenant trois instances de *str*. Le contenu d'un littéral de *list* n'a pas besoin d'être exprimé en littéral ; si les identifiants *a* et *b* ont été établis, alors la syntaxe [a, b] est légitime.

Le constructeur *list()* produit une *list* vide. Le constructeur accepte tout paramètre de type itérable : *str*, *list*, *tuple*, *set*, *dict*. Par exemple, *list(‘bonjour’)* produit une liste de caractères individuels, *[‘b’, ‘o’, ‘n’, ‘j’, ‘o’, ‘u’, ‘r’]*.

Une *list* existante est elle-même itérable, *tampon = list(data)* peut être utilisée pour construire une nouvelle instance de *list* référençant le même contenu que l'original.

tuple

Le *tuple* fournit une version immuable d'une séquence et, par conséquent, ses instances ont une représentation interne qui peut être plus simple que celle d'une liste.

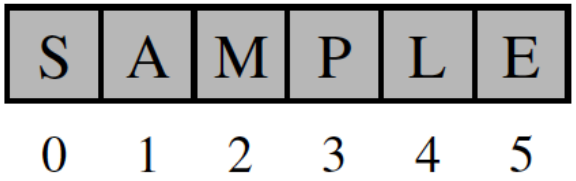
Alors que Python utilise les caractères [] pour délimiter une *list*, les parenthèses délimitent un *tuple*, avec () étant un tuple vide.

Il y a une subtilité importante. Pour exprimer un *tuple* de longueur 1 comme littéral, une virgule doit être placée après l'élément, mais entre parenthèses. Par exemple, (17,) est un *tuple* à un élément. La raison de cette exigence est que, sans la virgule, l'expression (17) est vue comme une simple expression numérique entre parenthèses.

str

La classe *str* de Python est spécifiquement conçue pour représenter efficacement une séquence de caractères immuable, basée sur le jeu de caractères international Unicode.

Les *str* ont une représentation interne plus compacte que les *lists* de référence et les *tuples* :



Une *str* Python est une séquence indexée de caractères.

Les littéraux de *str* peuvent être placés entre guillemets simples, comme dans ‘hello’, ou entre guillemets, comme dans “hello".

set

Le *set* représente la notion mathématique d'un ensemble, soit une collection d'éléments, sans doublons, et sans ordre.

Par opposition à une liste, le *set* dispose d'une méthode optimisée pour vérifier l'appartenance d'un élément spécifique.

```
djmaya2-iro:Mise à niveau major$ python setvslist.py
nb: 100000
temps pour construire la list: 0.002714872360229492 s
temps pour construire le set: 0.006640911102294922 s
temps pour chercher la list: 50.19043207168579 s
temps pour chercher le set: 0.008860111236572266 s
```

Le *set* utilise une structure de données connue sous le nom de table de hachage (que nous approfondirons).

Deux restrictions importantes :

i) l'ensemble ne maintient pas les éléments dans un ordre particulier ;

ii) seules les instances de types immuables peuvent être dans un *set* (e.g. *bool*, *int*, *float*, *tuple*, *str, frozenset*) ; mais pas de *lists* ni de *dict* ou *sets*, car ils sont mutables.

frozenset

Le *frozenset* est une forme immuable du *set*, donc il est légal d'avoir un *set* de *frozensets*.

Python utilise des accolades { et } comme délimiteurs pour un *set*, comme {17} ou {‘rouge’, ‘vert’, ‘bleu’}. L'exception à cette règle est que {} ne représente pas un *set* vide; pour des raisons historiques, il représente un *dict* vide (voir plus loin). Au lieu de cela, la syntaxe du constructeur *set*() produit un *set* vide.

Si un paramètre itératif est envoyé au constructeur, alors l'ensemble des éléments distincts est produit. Par exemple, *set(‘hello’)* produit {‘h’, ‘e’, ‘l’, ‘o’}.

dict

Le *dict* représente un dictionnaire ou la map d'un ensemble de clés distinctes à des valeurs associées. Par exemple, un *dict* peut correspondre à des numéros d'étudiant'es uniques, ou à toute information représentable par une instance immuable d'étudiant'es uniques.

Python implémente un *dict* en utilisant une approche presque identique à celle d'un *set* et store les valeurs associées.

Un *dict* utilise également des accolades. Comme le *dict* a été introduit dans Python avant le *set*, {} produit un *dict* vide.

Un *dict* non vide est exprimé avec une série de paires clé : valeur, séparées par des virgules : { 'ga': 'Irish', 'de': 'German' } fait correspondre 'ga' à irlandais et 'de' à allemand.

dict constructeur

Le constructeur de *dict* accepte un mappage existant comme paramètre, auquel cas il crée un nouveau *dict* avec les mêmes associations.

Alternativement, le constructeur accepte comme paramètre une paire de paires clé-valeur, comme dans
paires = [(ga, irlandais), (de, allemand)]
dict(paires)