Tranches

❶ Accès à un caractère par son indice
La notation `[i]` déjà rencontrée sur les chaines de caractères permet d'accéder au `i`-ème caractère d'une chaine où les caractères sont numérotés *à partir de 0*. Par exemple, si `exemple = "Un petit exemple"` :

U	n		p	e	t	i	t		e	x	e	m	p	l	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

alors : `exemple[0]` est `'U'`, `exemple[1]` est `'n'`, ...

❗ On remarquera que l'indice du dernier élément est *la longueur de la chaine moins 1*. La longueur s'obtenant avec `len`, ici on a par exemple `exemple[len(exemple)-1]` qui vaut `'e'`.

❷ Tranches
On peut aussi prendre une tranche en précisant dans les `[]` le début de la tranche (inclus) et sa fin (exclue) séparé par le caractère `:` si le début ou la fin sont absents alors ils correspondent respectivement au premier et au dernier indice. Par exemples :

- `exemple[3:8]` est `"petit"`
- `exemple[:2]` est `"Un"` (le début étant absent, on commence au premier caractère)
- `exemple[13:]` est `"ple"` (la fin étant absente, on termine au dernier caractère)

❸ Pas de progression
Une tranche peut prendre un troisième paramètre qui indique alors un *pas de progression*, par exemple si ce pas vaut 2, on ne prend qu'un caractère sur 2. D'autre part si le pas est négatif alors on progresse de la fin de la chaine vers le début. Par exemples :

- `exemple[4:10:2]` est `"ei "`
- `exemple[15:8:-1]` est `"elpmexe"`
- `exemple[::-1]` est `"elpmexe titep nU"` le pas étant négatif on progresse de la fin (absente donc dernier caractère) jusqu'au début (absent donc premier caractère).

Tuples

❶ Un tuple est une suite de valeurs repérées par leur indice (à la façon des caractères d'une chaine). Un tuple se note entre `()` et les valeurs sont séparées par des virgules. Par exemple `date = (2, "décembre", 1815)` est un tuple constituées de trois valeurs.

❷ On retrouve pour les tuples, la fonction `len`, l'accès au ième élément avec `[i]` et les tranches déjà vues sur les chaines de caractères.

❸ Les valeurs d'un tuple ne sont *pas modifiables* (comme les caractères d'une chaines), une tentative en ce sens produit un `TypeError`

❹ Un tuple peut être décompacté afin d'affecter chacune de ses valeurs à une variable. Par exemple `jour, mois, annee = date`.

Importation de fonctions

En Python, on peut importer des fonctions se trouvant dans d'autres modules, deux syntaxes sont possibles :

- `from <module> import <fonction>`, cela rend directement utilisable `<fonction>` dans la suite du programme. Par exemple la fonction racine carrée s'appelle `sqrt` et doit être importé depuis le module `math` avec `from math import sqrt` pour être utilisable.
- `import <module>`, dans ce cas, toutes les fonctions du module sont utilisables mais on doit préfixer leur nom par celui du module. Par exemple après un `import math` pour utiliser la fonction racine carrée, on doit écrire `math.sqrt`.

Listes

❶ Une liste est une suite de valeurs repérées par leur indice. Une liste se note entre `[` et `]` et les valeurs sont séparées par des virgules. Par exemple `premiers = [2, 3, 5, 7, 11, 13, 15]` est une liste. La liste vide est `[]`.

❷ On retrouve pour les listes, la fonction `len`, l'accès au ième élément avec `[i]` et les tranches déjà vues sur les chaines de caractères et les tuples.

❸ Les valeurs d'une liste, à la différence de celles d'un tuple,sont *modifiables*, on peut donc écrire `premiers[6]=17` afin que la liste ci-dessus devienne `premiers = [2, 3, 5, 7, 11, 13, 17]`.

❹ On peut ajouter un élément à une liste avec `append`, la syntaxe est `<liste>.append(<element>)`. Par exemple, après exécution de `premiers.append(19)` la liste ci-dessus devient `premiers = [2, 3, 5, 7, 11, 13, 17, 19]`.

❺ On peut retirer le dernier élément d'une liste avec `pop`, la syntaxe est `<liste>.pop()`. L'élément retiré est renvoyé par cette instruction et peut-être récupéré, ainsi `n = premiers.pop()` aura deux effets : supprimer 19 de la liste `premiers` et affecter cette valeur à `n`.

❻ Création de listes :

- en donnant explicitement ses éléments (comme la liste `premiers` ci-dessus).
- par répétition avec `*`, par exemple `[77]*10` est la liste constitué de 10 fois le nombre 77.
- par ajout successif, on part d'une liste vide et on ajoute (généralement à l'aide d'une boucle `for`) successivement avec `append` les éléments à la liste.
- par compréhension, à la façon dont on définit parfois les ensembles en mathématiques. Par exemple, `[i for i in range(50) if i%10==7]` est la liste `[7, 17, 27, 37, 47]` (les nombres entre 0 et 49 dont le reste dans la division euclidienne par 10 est 7).

Mutables et non mutables

En Python, certains types de données sont *mutables* et d'autres non, cela a des conséquences importantes lorsqu'on les manipule. Les listes de Python sont mutables au contraire de tous les autres rencontrés jusqu'ici (`int`, `float`, `bool`, `str`, `tuple`).

- Un type mutable est modifié lorsqu'on le passe en argument à une fonction pas un type non mutable.

Cas non mutable :

```
def incremente(n):
    n = n + 1

n= 42
incremente(n)
# n vaut toujours 42
```

Cas mutable :

```
def ajoute(x,l):
    l.append(x)

l = []
ajoute(2,l)
# l vaut [2]
```

- Si donne un autre nom à une variable mutable (avec `=`), toute modification de l'une des variables affecte aussi l'autre.

Cas non mutable :

```
n = 42
m = n
m = m + 1
# n vaut toujours 42
```

Cas mutable :

```
n = [12, 15]
m = n
m.append(17)
#n vaut maintenant [12, 15, 17]
```