

## T2.1 Compléments sur les listes

### 1. 2.1.9 Copie de listes

#### Vu en classe : une copie un peu trop parfaite

Observez le code ci-dessous, réalisé sans trucage.

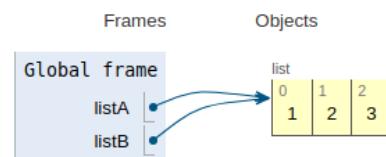
#### Script Python

```
>>> listA = [1, 2, 3]
>>> listB = listA
>>> listA.append(7)
>>> listB
[1, 2, 3, 7]
>>> listB.append(8)
>>> listA
[1, 2, 3, 7, 8]
```

Tout se passe comme si les listes `listA` et `listB` étaient devenus des clones «synchronisés» depuis l'affectation `listB = listA`.

#### Analyse grâce à PythonTutor

L'illustration de PythonTutor nous donne la clé de l'énigme :



`listA` et `listB` sont en fait **un seul et même objet**.

Comment en avoir le cœur net ? En observant leur adresse-mémoire, disponible grâce à la fonction `id` :

#### Script Python

```
>>> id(listA)
140485841327616
>>> id(listB)
140485841327616
```

Ceci met en évidence que la métaphore du tiroir dont on se sert pour expliquer ce qu'est une variable est malheureusement inexacte. Une variable est une référence vers une adresse-mémoire. Si deux variables font référence à la même adresse-mémoire, alors elles sont totalement identiques: toute modification de l'une entraîne une modification de l'autre.

## 1.1. Mais alors, comment copier le contenu d'une liste vers une autre sans créer un clone ?

### Deux façons (entre autres) de créer une vraie copie d'une liste

#### Script Python

```
>>> listA = [3, 4, 5]
>>> listB = listA.copy()
>>> listC = list(listA)
```

### Exercice 1

#### Énoncé

Contrôler les adresses mémoires avec la fonction `id` pour prouver que les exemples précédents produisent bien des objets différents.

#### Correction

## 2. 2.1.10 Listes en compréhension

### 2.1. Des exemples simples

On a déjà vu comment créer une liste *par extension*, c'est-à-dire en explicitant tous ses éléments:

#### Script Python

```
tab = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Ou bien par ajouts successifs à l'aide d'une boucle `for`, ce qui est bien plus efficace, surtout lorsque la taille du tableau est grande:

#### Script Python

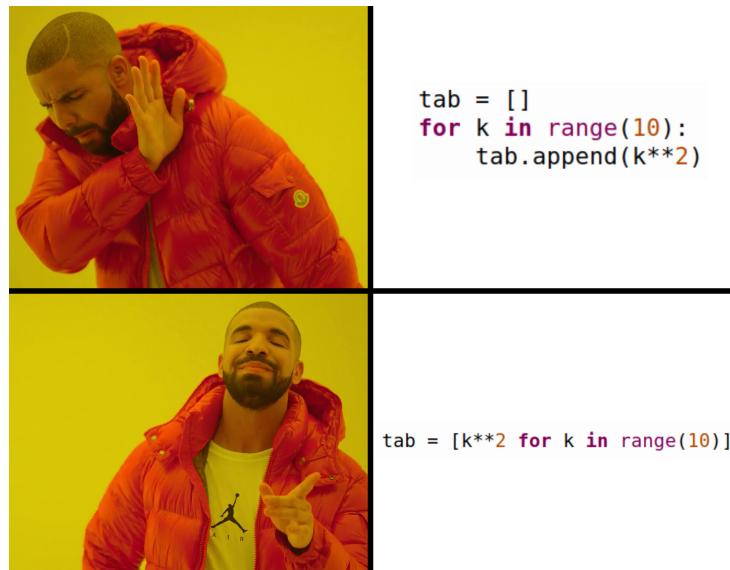
```
1 tab = []
2 for k in range(10):
3     tab.append(k**2)
```

Python permet également de combiner ces deux écritures, en «rentrant» la boucle `for` dans les crochets de définition de la liste:

#### Script Python

```
tab = [k**2 for k in range(10)]
```

On dit qu'on a créé la liste **en compréhension**: la liste des carrés des entiers allant de 0 à 9.



On peut également créer une liste en compréhension en parcourant les éléments d'un iterable déjà existant, au lieu d'un `range`. Créons par exemple la liste des images par une fonction `f` calculées sur une liste d'antécédents:

#### Script Python

```
1 antecedents = [-1, 0, 5, 10, 100]
2 def f(x):
```

```

4     return 2*x + 3
5
6 images = [f(x) for x in antecedents]

```

## 2.2. Avec un filtre

Reprendons par exemple l'exercice 4 du cours sur les listes. On souhaitait garder les valeurs positives de la liste `temp`:

### Script Python

```

1 temp = [11, 28, -16, -18, -10, 16, 10, 16, 2, 7, 23, 22, -4, -2, 19, 16, 22, -8, 18, -14, 29,
-1, 16, 22, -5, 6, 2, -4, 9, -17, -13, 22, 14, 24, 22, -9, -18, -9, 25, -11, 17, 17, 25, -10, 2,
-18, 29, 14, -16, 7]

```

La solution retenue:

### Script Python

```

1 temp_pos = []
2 for t in temp:
3     if t >= 0:
4         temp_pos.append(t)

```

On peut créer la même chose en compréhension, en incluant l'instruction conditionnelle `if` dans la définition de la liste:

### Script Python

```

1 temp_pos = [t for t in temp if t >= 0]

```

## 2.3. En deux dimensions

On peut se servir de cette méthode pour construire rapidement des tableaux à deux dimensions. Par exemple, voici comment on peut créer un tableau de zéros de 3 lignes et 5 colonnes:

### Script Python

```

tab = [5 * [0] for i in range(3)]

```

## 3. 2.1.11 Exercices

Dans chaque exercice, la liste doit être créée *en compréhension*.

## ☰ Exercice 1

Énoncé

Créer la liste constituée des valeurs absolues des entiers contenus dans la liste `valeurs`.

### ❖ Script Python

```
valeurs = [-2, 5, 1, -9, 2, 12, -8, -15, 7, 14, -27, 0, -2, 4, -5]
```

Correction

## ☰ Exercice 2

Énoncé

Créer la liste contenant tous les entiers inférieurs ou égaux à 100 multiples de 7.

Correction

## ☰ Exercice 3

Énoncé

Reprendre le pydéfi Le jardin des Hespérides en écrivant en compréhension la liste des nombres à sommer.

Correction

## Exercice 4

### Énoncé

On considère la liste suivante:

#### Script Python

```
lst = [51, 52, 66, 91, 92, 82, 65, 53, 86, 42, 79, 95]
```

Seuls les nombres entre 65 et 90 ont une signification : ce sont des codes Unicode de lettres (récupérables par la fonction `chr`).

Créer une liste `sol` qui contient les lettres correspondants aux nombres ayant une signification.

### Correction

## Exercice 5

### Énoncé

Consulter l'énoncé du pydéfi Le lion de Némée .

- Écrire une fonction prenant en paramètre une lettre et qui renvoie sa «valeur». Pour rappel:

#### Script Python

```
>>> ord('A')
65
```

- Écrire une fonction prenant en paramètre une chaîne de caractères et qui renvoie sa «valeur». Vous devez créer une liste en compréhension utilisant la fonction de la question 1, et l'utilisation de la fonction `sum` est autorisée.
- Créer en compréhension la liste des valeurs des divinités.

Pour la fonction `split` utilisée ci-dessous, voir sur cette page .

#### Script Python

```
1 | divinites = 'ARTEMIS ASCLEPIOS ATHENA ATLAS CHARON CHIRON CRONOS DEMETER EOS
  ERIS EROS GAIA HADES HECATE HEPHAISTOS HERA HERMES HESTIA HYGIE LETO MAIA
  METIS MNEMOSYNE NYX OCEANOS OURANOS PAN PERSEPHONE POSEIDON RHADAMANTHE
  SELENE THEMIS THETIS TRITON ZEUS'.split()
```

### Correction

#### Script Python

```
1 | divinites = 'ARTEMIS ASCLEPIOS ATHENA ATLAS CHARON CHIRON CRONOS DEMETER EOS
  ERIS EROS GAIA HADES HECATE HEPHAISTOS HERA HERMES HESTIA HYGIE LETO MAIA
  METIS MNEMOSYNE NYX OCEANOS OURANOS PAN PERSEPHONE POSEIDON RHADAMANTHE
  SELENE THEMIS THETIS TRITON ZEUS'.split()
5 |
6 | def valeur(lettre: str) -> int:
7 |     """
8 |         Renvoie la valeur d'une lettre capitale de l'alphabet.
9 |         Par ex: A -> 1, B -> 2, ... Z -> 26
10|        """
11|        return ord(lettre) - 64
12|
13| def valeur_mot(mot: str) -> int:
```

```
14     """
15     Renvoie la valeur d'un mot étant la somme des valeurs des lettres le
16     constituant.
17     """
18     valeurs_lettres = [valeur(l) for l in mot]
19     return sum(valeurs_lettres)

20     valeurs_divinites = [valeur_mot(d) for d in divinites]
```