T2.1 Compléments sur les listes

2.1.9 Copie de listes

!!! danger “Vu en classe : une copie un peu trop parfaite” Observez le code ci-dessous, réalisé sans trucage. `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`.

!!! aide “Analyse grâce à PythonTutor”

L’illustration de PythonTutor nous donne la clé de l’énigme : image{: .center width=30%}

`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` :

```
>>> 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.

Mais alors, comment copier le contenu d’une liste vers une autre sans créer un clone ?

!!! note “Deux façons (entre autres) de créer une vraie copie d’une liste” `python >>> listA = [3, 4, 5] >>> listB = listA.copy() >>> listC = list(listA)`

{{ initexo(0) }}

!!! example “{{ exercice() }}” === “Énoncé” Contrôler les adresses mémoires avec la fonction `id` pour prouver que les exemples précédents produisent bien des objets différents. === “Correction” {{ correction(False, " `python >>> listA = [3, 4, 5] >>> listB = list(listA) >>> listA.append(9) >>> listB [3, 4, 5] >>> id(listA) 140157471522368 >>> id(listB) 140157465797184 ` ") }}

2.1.10 Listes en compréhension

Des exemples simples

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

```
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 est bien plus efficace, surtout lorsque la taille du tableau est grande:

```
python linenums='1' tab = [] for k in range(10):     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:

```
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.

{: .center width=50%}

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:

“‘python linenums=‘1’ antecedents = [-1, 0, 5, 10, 100]

def f(x): return 2*x + 3

images = [f(x) for x in antecedents]

```
### Avec un filtre

Reprenons par exemple l'exercice 4 du cours sur les listes. On souhaitait garder les valeurs

```python linenums='1'
temp = [11, 28, -16, -18, -10, 16, 10, 16, 2, 7, 23, 22, -4, -2, 19, 16, 22, -8, 18, -14, 29
```

La solution retenue: `python linenums='1' temp_pos = [] for t in temp:`
`if t >= 0:          temp_pos.append(t)`

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

```
python linenums='1' temp_pos = [t for t in temp if t >= 0]
```

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:
```

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

2.1.11 Exercices

{{ initexo(0) }}

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

!!! example "{{ exercice() }}" === "Énoncé" Créer la liste constituée des valeurs absolues des entiers contenus dans la liste `valeurs`.

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

=== "Correction"
    {{ correction(False,
    "
    ```python linenums='1'
    valeurs = [-2, 5, 1, -9, 2, 12, -8, -15, 7, 14, -27, 0, -2, 4, -5]
    absolues = [abs(v) for v in valeurs]
    ```

    "
) }}

!!! example "{{ exercice() }}" === "Énoncé" Créer la liste contenant tous les entiers inférieurs ou égaux à 100 multiples de 7. === "Correction" {{ correction(False, " python linenums='1' lst = [k for k in range(101) if k%7 == 0]

    "
) }}

!!! example "{{ exercice() }}" === "Énoncé" Reprende le pydéfi Le jardin des Hespérides{:target="_blank"} en écrivant en compréhension la liste des nombres à sommer.

=== "Correction"
    {{ correction(False,
    "
    ```python linenums='1'
    pommes = [etage ** 2 for etage in range(51) if etage%3 == 0]
    ```

    "
) }}

!!! example "{{ exercice() }}" === "Énoncé" On considère la liste suivante: 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 sign
=== "Correction"
    {{ correction(False,
    "
    ```python linenums='1'
    sol = [chr(code) for code in lst if code >= 65 and code <= 90]
    ```

    "
) }}
```

!!! example "{{ exercice() }}" === "Énoncé" Consulter l'énoncé du pydéfi Le lion de Némée{:target="_blank"} .

```
    1. Écrire une fonction prenant en paramètre une lettre et qui renvoie sa «valeur». Pour
    ```python
    >>> ord('A')
    65
    ```

    2. Écrire une fonction prenant en paramètre une chaîne de caractères et qui renvoie sa <
    3. Créer en compréhension la liste des valeurs des divinités.

    Pour la fonction `split` utilisée ci-dessous, voir [sur cette page](https://cgouygou.git

    ```python linenums='1'
    divinites = 'ARTEMIS ASCLEPIOS ATHENA ATLAS CHARON CHIRON CRONOS DEMETER EOS ERIS EROS C
    ```


=== "Correction"
    {{ correction(Trye,
    "
    ```python linenums='1'
    divinites = 'ARTEMIS ASCLEPIOS ATHENA ATLAS CHARON CHIRON CRONOS DEMETER EOS ERIS EROS C

    def valeur(lettre: str) -> int:
        '''
        Renvoie la valeur d'une lettre capitale de l'alphabet.
        Par ex: A -> 1, B -> 2, ... Z -> 26
        '''
        return ord(lettre) - 64

    def valeur_mot(mot: str) -> int:
        '''
```
```

```
    Renvoie la valeur d'un mot étant la somme des valeurs des lettres le
    constituant.
    '''
    valeurs_lettres = [valeur(l) for l in mot]
    return sum(valeurs_lettres)

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

) }}