transcodage_while_lists_26

October 24, 2025

[4]:
```
lnb = [9,3,5,7]
```

Supposons que je veuille afficher le 3e élément de cette liste

[5]:
```
print(lnb[2])
```

```
5
```

[ ]:
```
Supposons maintenant que je veuille afficher tous les éléments de la liste:
```

[7]:
```
print(lnb[0])
print(lnb[1])
print(lnb[2])
print(lnb[3])
```

```
9
3
5
7
```

Les dépassements d'index sont gérés par le mécanisme interne de Python

[8]:
```
print(lnb[4])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[8], line 1
----> 1 print(lnb[4])

IndexError: list index out of range
```

ça marche aussi avec des liste d'autres objets, par exemple liste hétérogène ou chaîne de caractères vue comme une liste

[9]:
```
lh = [5, False, "a", 7.2, [1,2,3]]
print(lh[0])
print(lh[1])
print(lh[2])
```

```
5
False
a
```

[10]:
```
chaine = "maison"
print(chaine[0])
print(chaine[1])
print(chaine[2])
```

```
m
a
i
```

Limite évidente: bcp d'instructions à écrire pour des listes qui peuvent être très longues. Il faut trouver un moyen de faciliter l'écriture de la répétition.

Utilisons une variable "index". Attention à faire la mise à jour de l'index à chaque tour de boucle. La version sans mise à jour boucle indéfiniment. (démo sur codepad)

Avec la mise à jour de l'index:

[11]:
```
index = 0
while (index < 4):
    print(lnb[index])
    index += 1
```

```
9
3
5
7
```

Ici on a écrit "en dur" le nombre d'éléments de la liste. Mais c'est un point de fragilité: d'une part la liste a pu être mise à jour avec possiblement une augmentation du nb d'éléments, et d'autre part on peut ne pas connaître à l'avance le nombre d'éléments de la liste. Solution : utiliser la méthode len()

[12]:
```
index = 0
while (index < len(lnb)):
    print(lnb[index])
    index += 1
```

```
9
3
5
7
```

[13]:
```
lnb.append(1)
```

[14]:
```
index = 0
while (index < len(lnb)):
    print(lnb[index])
```

```
    index += 1
```

```
9
3
5
7
1
```

boucle for: une simplification de l'usage du while

[8]:
```
lnb = [6,7,4,5,8]
i = 0
while (i < len(lnb)):
    print(lnb[i], end = ' ')
    i += 1
print()

for i in [0,1,2,3,4]:
    print(lnb[i], end = ' ')
print()

for i in range(len(lnb)):
    print(lnb[i], end = ' ')
print()

for n in lnb:
    print(n, end = ' ')
```

```
6 7 4 5 8
6 7 4 5 8
6 7 4 5 8
6 7 4 5 8
```

1 TP pour séance du 24 octobre:

1.1 Première série

1. La variable cible contient un mot de votre choix. Ecrire une boucle qui affiche une à une toutes les lettres du mot en remplaçant les *e* minuscule par un *3* (les autres lettres sont inchangées).
2. Ecrire une boucle qui remplace toutes les voyelles par la lettre *x*
3. Ecrire une boucle qui remplace une lettre sur deux par le symbole '_' (underscore)
4. Ecrire un programme qui demande à l'utilisateur un mot, et fait la transformation précédente.

1.2 Deuxième série

1. On suppose qu'on a une liste de "correspondances' ': chaque correspondance est une liste de deux lettres, par exemple `["e","3"]`. Demander à l'utilisateur une lettre, et indiquer (print) si cette lettre à un correspondant dans la liste.

2. Etant donnée une correspondance telle que définie à la question précédente, demander un mot à l'utilisateur, et afficher ce mot lettre par lettre en appliquant les remplacements indiqués dans la liste de correspondance.
3. A partir de deux mots de même longueur, construire une correspondance telle que définie à la question 1. Quelles précautions doit-on prendre ?
4. Demander deux mots à l'utilisateur, construire une correspondance à partir de ces deux mots; demander ensuite un nouveau mot, qui sera affiché en appliquant la correspondance.
5. Faire en sorte que l'utilisateur puisse répéter l'opération jusqu'à ce qu'il saisisse le mot `fin`.