1ère NSI — Quiz 03 : listes & chaines de caractères

Question 1 – manipulation de liste

```
1 nombres = [2, 4, 6, 8]
2 nombres[1] += nombres[1]
3 print(nombres)
```

Qu'affichera l'exécution de ce script ?

Aucun piège ici : la deuxième ligne augmente ("+=") l'élément d'indice 1 de la liste de lui-même, donc ici 4 de 4 (puisque les indices commencent à 0). L'affichage sera donc : `[2, 8, 6, 8]`

Question 2 – comportement listes et chaines

```
1 lst = ["a", "b", "c"]
2 chn = "abc"
3 print(lst[1])
4 print(chn[2])
5 lst[1] = "ZZ"
6 print(lst[1])
7 chn[2] = "ZZ"
8 print(chn[2])
```

Si j'execute ce code en console (donc une ligne après l'autre, en poursuivant même si une erreur est rencontrée), qu'est-ce qui va s'afficher à l'écran (il n'y a qu'un seul piège...) ?

L'affichage sera, pour chaque ligne :

1. *Rien (c'est juste une affectation).*
2. *Rien (c'est juste une affectation).*
3. *Élément d'indice 1 de la liste : `b`.*
4. *Élément d'indice 2 de la chaine : `c`.*
5. *Rien (c'est juste une affectation d'une valeur à un élément de la liste).*
6. *La nouvelle valeur de l'élément d'indice 1 de la liste : `ZZ`.*
7. *(Il est là, le piège...) Les chaines de caractère sont ce qu'on appelle "immutables" – il n'est pas possible d'en modifier directement les éléments, il faut passer si on souhaite faire ça par une autre variable. Donc l'opération demandée ici est impossible en Python et on obtiendra donc un message d'erreur "`TypeError: 'str' object does not support item assignment`".*
8. *L'opération précédente s'étant soldée par une erreur (et comme on est ici en console, c'est-à-dire qu'on exécute toutes les lignes du code même si on rencontre une erreur), on affichera ici l'élément d'indice 2 de la chaine qui n'a pas changé : `c`.*

Question 3 – une syntaxe qu'on a vue – mais très rapidement...

```
1 declaration = "Python est amusant. J'ai envie d'en apprendre plus."
2 phrases = declaration.split(".")
3 print(phrases[0])
4 mots = declaration.split()
5 print(mots[4])
```

Qu'affichera ce code ? (essayez de deviner même si vous ne vous souvenez pas de la méthode "split" !)

Un but de cette question était de vous convaincre que même si vous ne vous souvenez pas d'une syntaxe spécifique vous pouvez tout à fait la retrouver simplement en lisant et en vous forçant à vous plonger dans le code…

"Split", comme son nom l'indique, découpe une chaine de caractères selon un séparateur et en met les éléments dans une liste.

Donc le premier split va découper selon le séparateur "." et générer la liste suivante : `['Python est amusant', " J'ai envie d'en apprendre plus", '']` (notez la phrase vide à la fin – puisque la déclaration se termine par un point)

Donc l'affichage de la ligne 3 donnera `Python est amusant`

Le second split, lui, n'a pas de séparateur spécifié – et par défaut, ce sera l'espace qui sera utilisé (ce que le nom de la liste, "mots", aurait dû vous permettre de deviner). Donc il génèrera la liste suivante : `['Python', 'est', 'amusant.', "J'ai", 'envie', "d'en", 'apprendre', 'plus.']`

Donc l'affichage de la ligne 5 donnera `envie`

Question 4 – message d'erreur

```
1 age = 16
2 nom = "Sophia"
3 message = "Je m'appelle " + nom + "et j'ai " + age + " ans"
```

J'ai cherché à exécuter ce script et j'ai obtenu l'erreur suivante. Quel veut-elle dire ?

```
Traceback (most recent call last):
File "C:\Users\Marc\PyProj\exemple.py", line 3, in <module>
message = "Je m'appelle " + nom + "et j'ai " + age + " ans"
TypeError: can only concatenate str (not "int") to str
```

Comme toujours, un message d'erreur se lit de bas en haut : à la fin on voit la nature de l'erreur (`"TypeError"`), au-dessus l'instruction qui a posé problème (`"message = ..."`), et encore au-dessus la localisation de cette instruction (dans ce cas la $3^{ème}$ ligne du fichier exemple.py).

En l'occurrence l'erreur vient du fait qu'on a cherché à concaténer (mettre bout à bout) des chaînes de caractères avec un entier – ici, age. Pour remédier à cela il faudrait mettre `str(age)` dans l'affectation de la variable message.