Semaine du 7 au 11 décembre

Dans l'ensemble, le travail à la maison a été fait de manière trop superficielle. Il est absolument **fondamental** de travailler ce qui vous est demandé d'une semaine sur l'autre, au risque de faire du sur-place. C'est ce qui est arrivé pour la première séance de la semaine.

Pour que nous fassions un peu de programmation Python, je vous propose de travailler sur un mini-projet autour de la notion d'algorithme et de tableau bi-dimensionnel (comme dans l'exercice 6 de la feuille d'exercices).

Vous trouverez l'énoncé sur Pronote ainsi qu'à cette adresse : https://urlz.fr/evmO

Faites ce travail en essayant de toujours revenir à la notion d'algorithme avant de coder en Python. Cela vous aidera beaucoup.

Séance 1 (mardi 15 décembre ou jeudi 16 décembre) :

Retour sur les calculs de complexité : vérifiez bien que vous êtes capable de calculer la complexité d'un algorithme donné en comptant le nombre d'opérations élémentaires ainsi que le nombre de boucles imbriquées.

Exercice 4 :
Nous avons commencé l'exercice sur le bi-gramme.
mot : ALLIGATOR
lettre1 : A
lettre2 : R
A, R est un bi-gramme

lettre1 : I
lettre2 : L
I, L n'est pas un bi-gramme car le I n'est jamais suivi d'un R.

lettre1 : T
lettre2 : Z
T, Z n'est pas un bi-gramme car le T n'est jamais suivi d'un Z (il n'est même pas dedans!).

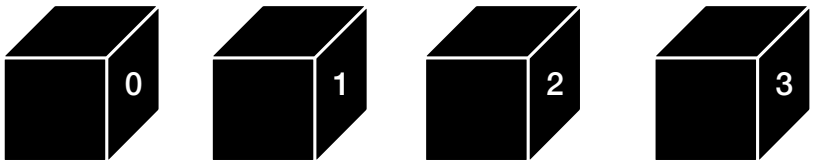
Donc mes données sont :
mot : le mot que l'on considère
lettre1 : la première lettre du bigramme
lettre2 : la deuxième lettre du bigramme
reponse : booléen m'indiquant si j'ai un bi-gramme ou non
reponse1 : booléen m'indiquant si j'ai trouvé la 1ère lettre

Correction page suivante…

Correction page suivante

Exercice 3 :

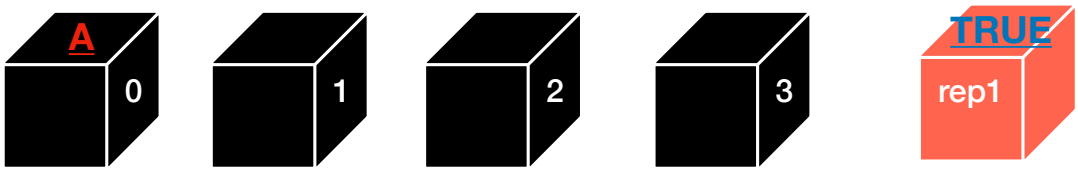
a) Pour **trouver un bi-gramme**, encore une fois, réfléchissez en terme de cases-mémoire dans l'ordinateur (sur un exemple, disons ALLIGATOR, letre1 = A, lettre2 = L) :



Encore une fois, il va donc falloir faire une boucle : n'ayez pas peur et **ÉCRIVEZ CETTE BOUCLE**

```
pour i allant 0 à la longueur du mot :
```

On va donc devoir ouvrir chaque boite l'une après l'autre :

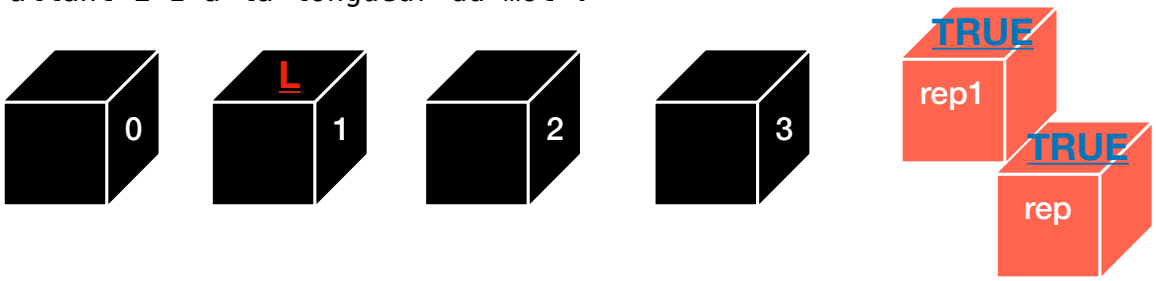


numéro 0 : Ok, est-ce que cette lettre n°0 est un A ? oui ! Donc, enregistrement dans reponse1.
Ca nous donne dans l'algorithme :

```
      si la lettre numéro i == lettre1, alors : rep1 = TRUE
```

Bon, ici, il faut l'idée : on a trouvé le A, on va chercher un L dans le reste du mot (donc, dans les lettres n°i+1 et supérieures). En terme algorithmique, on va ouvrir chaque boite l'une après l'autre depuis la i+1-ième jusqu'à la dernière.

```
      pour j allant i+1 à la longueur du mot :
```



numéro 1 : On teste, est-ce que la lettre n°1 est un L ? oui ! Donc, réponse passe à TRUE.
On a donc trouvé les deux lettres du bi-gramme dans le bon ordre.

```
            si la lettre numéro j == lettre2, alors : reponse = TRUE
```

Algorithme complet :

```
pour i allant 0 à la longueur du mot :
      si la lettre numéro i == lettre1, alors :
            rep1 = TRUE
            pour j allant i+1 à la longueur du mot :
                  si la lettre numéro j == lettre2, alors :
                  reponse = TRUE
```

Complexité ??

Correction page suivante

Bon, la complexité est en $\mathcal{O}(n^2)$ car les deux boucles sont imbriquées et dépendent toutes les deux de n. D'ailleurs le pire des cas est étonnant…

Il correspondrait à rechercher le bigramme A,A dans le mot AAAAAAAAAAAAAAAAAAAA !

On peut faire bien mieux :

Si on a trouvé la lettre1 du bi-gramme, la variable rep1 est passé à TRUE. On sait donc qu'on l'a trouvé. Nous pouvons donc continuer et accepter la lettre2 du bi-gramme uniquement si c'est la bonne lettre (normal quoi…) et aussi si la lettre1 du bi-gramme a été trouvé !

À vous d'écrire ce nouvel algorithme qui doit être de complexité linéaire…

Correction page suivante