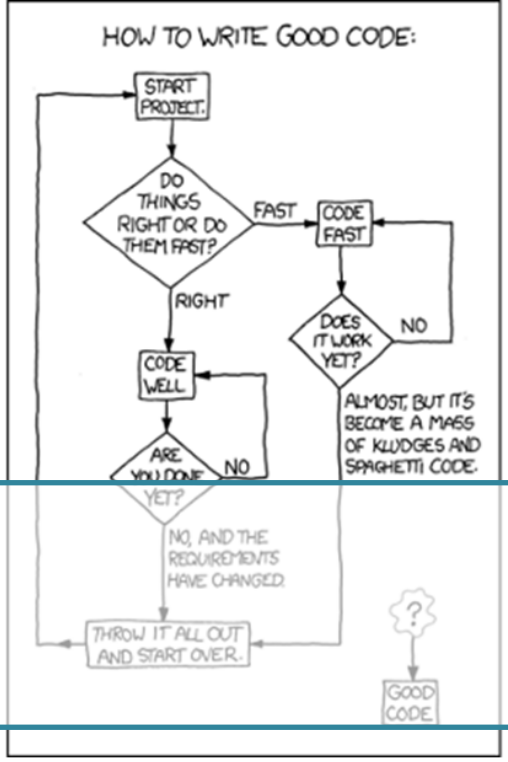
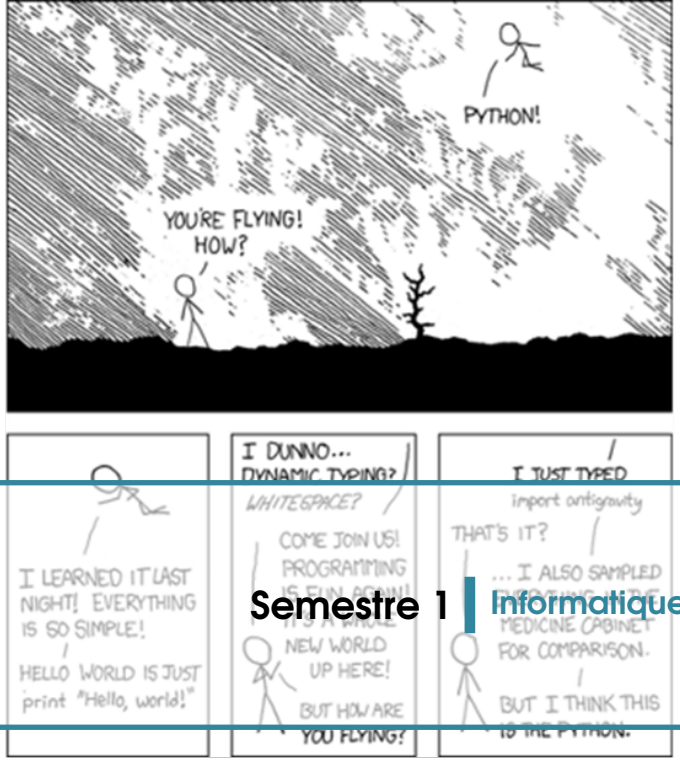




Thèmes d'étude

Recherche séquentielle

Thème :

- Recherche séquentielle dans un tableau unidimensionnel. Dictionnaire.

Exemple d'activités :

- Recherche d'un élément.
- Recherche du maximum, du second maximum.
- Comptage des éléments d'un tableau à l'aide d'un dictionnaire.

Recherche séquentielle

Exercice 1 – Exercices d'échauffement

Objectif Rechercher séquentiellement un élément dans un tableau unidimensionnel ou dans un dictionnaire.

Recherche d'un nombre dans une liste

Nous allons commencer par rechercher si un nombre est dans un tableau.

Commençons par définir la liste des entiers pairs compris entre 0 et `nb` exclus.

```
def generate_pair_01(nb: int) -> list :
    """
    Génération d'une liste de nombres pairs compris entre 0 (exclus) et nb (exclus).
    """
    res = []
    for i in range(1,nb//2):
        res.append(2*i)
    return res
```

Recopier la fonction dans un fichier.

Question 1 *Vérifier que la fonction* `generate_pair_01` *fonctionne pour* `nb=0`, `nb=9`, `nb=10`.

Question 2 *Écrire une fonction de signature* `generate_pair_02(nb: int) -> list` *en utilisant une boucle* `while`.

Question 3 *Écrire une fonction de signature* `recherche_nb_01(nb: int, L: list) -> bool` *qui renvoie* `True` *si* `nb` *est dans* `L`, `False` *sinon. On utilisera une boucle* `for`.

Question 4 *Écrire une fonction de signature* `recherche_nb_02(nb: int, L: list) -> bool` *qui renvoie* `True` *si* `nb` *est dans* `L`, `False` *sinon. On utilisera une boucle* `while`.

Question 5 *Écrire une fonction de signature* `recherche_nb_03(nb: int, L: list) -> bool` *qui renvoie* `True` *si* `nb` *est dans* `L`, `False` *sinon. On n'utilisera pas explicitement de boucles* `for` *ou* `while`.

Question 6 *Écrire une fonction de signature* `recherche_first_index_nb_01(nb: int, L: list) -> int` *qui renvoie l'index de la première appartion du nombre* `nb` *dans la liste* `L`. *La fonction renverra* `-1` *si* `nb` *n'est pas dans la liste. On utilisera une boucle* `for`.

Question 7 *Écrire une fonction de signature* `recherche_first_index_nb_02(nb: int, L: list) -> int` *qui renvoie l'index de la première appartion du nombre* `nb` *dans la liste* `L`. *La fonction renverra* `-1` *si* `nb` *n'est pas dans la liste. On utilisera une boucle* `while`.

Question 8 *Écrire une fonction de signature* `recherche_last_index_nb_01(nb: int, L: list) -> int` *qui renvoie l'index de la dernière appartion du nombre* `nb` *dans la liste* `L`. *La fonction renverra* `-1` *si* `nb` *n'est pas dans la liste. On utilisera une boucle* `for`.

Question 9 *Écrire une fonction de signature* `recherche_last_index_nb_02(nb: int, L: list) -> int` *qui renvoie l'index de la dernière appartion du nombre* `nb` *dans la liste* `L`. *La fonction renverra* `-1` *si* `nb` *n'est pas dans la liste. On utilisera une boucle* `while`.

Question 10 *Écrire une fonction de signature* `recherche_index_nb_01(nb: int, L: list) -> list` *qui renvoie la liste des index du nombre* `nb` *dans la liste* `L`. *La fonction renverra une liste vide si* `nb` *n'est pas dans la liste.*

Recherche d'un caractère dans une chaîne (de caractères)

Question 11 *Écrire une fonction de signature* `is_char_in_str_01(lettre: str, mot: str) -> int` *qui renvoie* `True` *si* `lettre` *est dans* `mot`, `False` *sinon. On utilisera une boucle* `for` *ou* `while`.

Question 12 *Écrire une fonction de signature* `is_char_in_str_02(lettre: str, mot: str) -> int` *qui renvoie* `True` *si* `lettre` *est dans* `mot`, `False` *sinon. On n'utilisera ni boucle* `for` *ni* `while` *explicite.*

Question 13 *Écrire une fonction de signature* `compte_lettre_01(lettre: str, mot: str) -> int` *qui renvoie le nombre d'occurrences de* `lettre` *dans le* `mot`.

Les instructions suivantes permettent de charger l'ensemble des mots du dictionnaire dans la variable `dictionnaire`. `dictionnaire` est une liste de mots. Chacune des lettres de l'alphabet sont stockées dans la variable `alphabet`.

```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
def load_fichier(file):
    fid = open(file,'r')
    mots = fid.readlines()
    fid.close()
    return mots
dictionnaire = load_file('liste_francais.txt')
```

Question 14 *Écrire une fonction de signature* `compte_lettre_02(lettre: str, mots: list) -> int` *qui renvoie le nombre d'occurrences de* `lettre` *dans une liste de mots* `mots`.

Question 15 *Quelle consonne apparaît le plus souvent ? Quelle consonne apparaît le moins souvent ? Indiquer le nombre d'occurrences dans chacun des mots*

Question 16 *Écrire une fonction de signature* `mots_plus_long(mots: list) -> str` *qui renvoie le mot le plus long.*

Question 17 *Écrire une fonction de signature* `cherche_mot_in_chaine_01(mot: str, chaine: str) -> int` *qui renvoie* `True` *si* `mot` *est dans* `chaine`, `False` *sinon. On utilisera des boucles* `for` *ou* `while`.

Question 18 *Écrire une fonction de signature* `cherche_mot_in_chaine_02(mot: str, chaine: str) -> int` *qui renvoie* `True` *si* `mot` *est dans* `chaine`, `False` *sinon. On n'utilisera ni boucle* `for` *ni* `while`.

Question 19 *Écrire une fonction de signature* `cherche_mot_in_dico(nb: int, dico: lst) -> str` *qui permet de trouver le mot de* `nb` *lettres qui est le plus contenu dans d'autres mots.*

Recherche dans un dictionnaire

Un dictionnaire (`dict`) est un type composite (au même titre que les chaînes, les listes ou les tuples). Les éléments d'un dictionnaire sont constitués d'une **clé** (alphabétique ou numérique par exemple) et d'une valeur. À la différence d'une liste par exemple, les éléments ne sont pas ordonnés.

▪ **Exemple** Dans le but de faire un comptage du nombre de lettres des mots du dictionnaire, nous allons créer un dictionnaire constitué des lettres de l'alphabet (clés) et de leur nombre d'apparitions (valeurs).

```
nb_lettres = {}
nb_lettres['a']=0
nb_lettres['b']=0
print(nb_lettres)
    {'a': 0, 'b': 0}
```

```
nb_lettres = {}
for lettre in alphabet :
    nb_lettres[lettre]=0
print(nb_lettres["a"])
    0
```

Tester l'appartenance d'une clé à un dictionnaire : `"a" in nb_lettres`.
Supprimer une clé d'un dictionnaire : `del nb_lettres["a"]`.
Parcourir un dictionnaire :

```
for clef in nb_lettres :
    print(clef)
    print(clef,nb_lettres[clef])
```

```
for clef,valeur in nb_lettres.items() :
    print(clef,valeur)
```

▪

Exercice 2 – La conjecture de Syracuse

D'après Jean-Pierre Becirspahic, `https://info-llg.fr/`

Objectif

- ...

On doit cette conjecture au mathématicien allemand Lothar Collatz qui, en 1937, proposa à la communauté mathématique le problème suivant : « on part d'un nombre entier strictement positif; s'il est pair on le divise par 2, s'il est impair on le multiplie par 3 et on ajoute 1. On réitère ensuite cette opération.»

Par exemple, à partir de 14 on construit la suite de nombres :

$$14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2...$$

Après que le nombre 1 ait été atteint, la suite des valeurs (1, 4, 2, 1, 4, 2 ...) se répète indéfiniment en un cycle de longueur 3.

La conjecture de Syracuse [1] est l'hypothèse mathématique selon laquelle n'importe quel entier de départ conduit à la valeur 1 au bout d'un certain temps.

Nous allons expérimenter cette conjecture en programmant l'évolution de la suite $(u_n)_{n\in\mathbb{N}}$ définie par les relations :

$$u_0 = c \text{ et } \forall n \geq 1, u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}.$$

Temps de vol et altitude maximale

Le temps de vol d'un entier c et le plus petit entier n (en admettant qu'il existe) pour lequel $u_n = 1$. Par exemple, le temps de vol pour $c = 14$ est égal à 17.

Question 1 *Définir une fonction nommée* `tempsdevol` *prenant un paramètre entier* `c` *et retournant le plus petit entier n pour lequel $u_n = 1$.*

De manière tout aussi imagée, on appelle altitude maximale de c la valeur maximale de la suite $(u_n)_{n\in\mathbb{N}}$. Par exemple, l'altitude maximale de $c = 14$ est égale à 52.

Question 2 *Modifier votre algorithme pour définir une fonction nommée* `altitude` *qui calcule cette fois-ci l'altitude maximale pour un entier c donné en paramètre.*

Vérification expérimentale de la conjecture

On désire désormais vérifier la validité de la conjecture pour toute valeur $c \leq 1\,000\,000$. Une première solution consisterait à calculer le temps de vol pour toutes ces valeurs, mais ce calcul est long et il y a mieux à faire en observant que si la conjecture a déjà été vérifiée pour toute valeur $c' < c$, il suffit qu'il existe un rang n pour lequel $u_n < c$ pour être certain que la conjecture sera aussi vérifiée au rang c.

On appelle temps d'arrêt (ou encore temps de vol en altitude) le premier entier n (s'il existe) pour lequel $u_n < c$.

Question 3 *Écrire une fonction* `tempsdarret` *prenant un paramètre entier c et retournant le temps d'arrêt de la suite de Syracuse correspondante.*

Nous souhaitons maintenant mesurer le temps nécessaire pour vérifier la conjecture jusqu'à un paramètre entier m. Pour cela, nous allons utiliser la fonction `time` du module time du même nom, sans argument, qui retourne le temps en secondes depuis une date de référence (qui dépend du système).

Question 4 *À l'aide de cette fonction écrire une fonction* `verification` *qui prend en argument un paramètre entier m et retourne le temps nécessaire pour vérifier que toutes les valeurs $c \in [\![2, m]\!]$ ont bien un temps d'arrêt fini. Quelle durée d'exécution obtient-t-on pour $m = 1\,000\,000$?*

Question 5 *Quel est le temps d'arrêt d'un entier pair ? et d'un entier de la forme $c = 4n + 1$? En déduire qu'on peut restreindre la recherche aux entiers de la forme $4n + 3$, et modifier en conséquence la fonction précédente. Combien*

1. Du nom de l'université américaine qui a popularisé ce problème.

de temps gagne-t-on par rapport à la version précédente pour $m = 1\,000\,000$? Vérifier ensuite la conjecture pour $m = 10\,000\,000$.

Records

Question 6 *Déterminer l'altitude maximale que l'on peut atteindre lorsque $c \in [\![1, 1\,000\,000]\!]$, ainsi que la valeur minimale de c permettant d'obtenir cette altitude.*

Question 7 *Déterminer le temps de vol en altitude (autrement dit le temps d'arrêt) de durée maximale lorsque $c \in [\![1, 1\,000\,000]\!]$ ainsi que la valeur de c correspondante.*

Question 8 *On appelle vol en altitude de durée record un vol dont tous les temps d'arrêt de rangs inférieurs sont plus courts. Par exemple, le vol réalisé pour $c = 7$ est un vol en altitude de durée record (égale à 11) car tous les vols débutant par $c = 1, 2, 3, 4, 5, 6$ ont des temps d'arrêt de durées inférieures à 11. Déterminez tous les vols en altitude de durée record pour $c \leq 1\,000\,000$.*

Affichage du vol

Pour obtenir des graphes, on utilise la fonction plot qui appartient à un module appelé `matplotlib.pyplot` et dédié au tracé de graphes. Vous allez donc commencer par importer celui-ci à l'aide de la commande : `import matplotlib.pyplot as plt` Désormais toutes les fonctions de ce module vous sont accessibles à condition de les préfixer par `plt`. Nous découvrirons progressivement les nombreuses possibilités qu'offre ce module, mais aujourd'hui nous n'aurons besoin que de deux fonctions : `plt.plot` et `plt.show`. Sous sa forme la plus simple, la fonction `plt.plot` n'exige qu'une liste en paramètre : `plt.plot([a0, a1, ..., an])` crée un graphe constitué d'une ligne brisée reliant les points de coordonnées (k, a_k) pour $k \in [\![0, n]\!]$. En Python, une liste est encadrée par des crochets et ses éléments séparés par une virgule. Nous étudierons les listes plus tard dans le cours ; pour l'instant nous n'aurons besoin que du résultat suivant : si `lst` est une liste, on ajoute un élément `x` à celle-ci à l'aide de la commande : `lst.append(x)`. Une fois votre graphe créé par la fonction `plt.plot`, il reste à le faire apparaître dans une fenêtre annexe à l'aide de l'instruction `plt.show()`.

Question 9 *Définir une fonction nommée graphique qui prend un entier c en paramètre et qui construit le graphe de la suite $(u_n)_{n\in\mathbb{N}}$ durant son temps de vol.*

Structures imbriquées

Exercice 3 – Trouver les deux valeurs les plus proches dans un tableau

Question 1 *On se donne un tableau* `T` *unidimensionnel. Écrire une fonction* `distance_min1(T)` *qui renvoie les deux éléments qui sont les plus proches ie dont la valeur absolue de la différence est minimale. On indiquera les valeurs obtenues ainsi que les indices correspondants.*

Question 2 *Pour un tableau à n cases, montrer que le nombre de comparaisons $C(n)$ faites dans cette fonction est tel que la suite $\left(\frac{C(n)}{n^2}\right)$ est bornée : on dit que la* ***complexité est quadratique****.*

Question 3 *Faire la même question pour un tableau bidimensionnel en écrivant une fonction* `distance_min2(T)`*. Ici,* `T` *sera donc une matrice pas nécessairement carrée par exemple de la forme*

```
T=[[1,2,3],[6,4,3],[3,8,9],[3,-2,0]]:
```

chaque élément de `T` *désignera une ligne du tableau. Le nombre de ligne est le nombre d'éléments de* `T`*, le nombre de colonnes est le nombre d'éléments de* `T[0]`*.*

Exercice 4 – Recherche d'un mot dans un texte

Question 1 *Écrire une fonction* `est_ici(texte,motif)` *qui, étant données deux chaines de caractères* `texte` *et* `motif`*, renvoie* `True` *ou* `False` *selon que mot est ou n'est pas dans texte. On n'utilisera pas de slicing mais on fera deux méthodes, l'une sans booléen explicite, l'autre avec.*

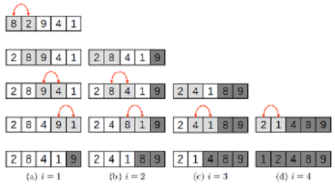
Question 2 *Écrire une fonction* `est_sous_mot(texte,motif)` *qui renvoie* `True` *ou* `False` *selon que motif est dans texte ou pas.*

Question 3 *Écrire une fonction* `position_sous_mot(texte,motif)` *qui renvoie la liste de toutes les occurences de l'indice de position du mot* `motif` *dans* `texte`*.*

Exercice 5 – Tri à bulles

Le tri à bulles est un algorithme de tri classique. Son principe est simple, et il est très facile à implémenter. On considère un tableau de nombres T, de taille N. L'algorithme parcourt le tableau, et dès que deux éléments consécutifs ne sont pas ordonnés, les échange. Après un premier passage, on voit que le plus grand élément se situe bien en dernière position. On peut donc recommencer un tel passage, en s'arrêtant à l'avant-dernier élément, et ainsi de suite.

Au i-ème passage on fait remonter le i-ème plus grand élément du tableau à sa position définitive, un peu à la manière de bulles qu'on ferait remonter à la surface d'un liquide, d'où le nom d'algorithme de tri à bulles.



Question 1 *Appliquer l'algorithme de tri à Bulles «à la main» au tableau ci-dessous :*

2	1	6	9	8	4

Question 2 *Écrire une fonction* `est_trie(T)` *qui renvoie* `True` *ou* `False` *selon que le tableau* `T` *est trié ou pas.*

Question 3 *Écrire une fonction* `TriBulles(T)` *triant le tableau* `T` *par l'algorithme de tri à bulles*[2]

Question 4 *Vérifier que la fonction ci-dessus est correcte en utilisant la fonction* `EstTrie` *sur des tableaux de nombres aléatoires. On rappelle que la bibliothèque random permet de créer des nombres aléatoires.* `random.randint(a,b)` *renvoie un entier aléatoire compris entre a et b.*

Question 5 *Montrer que la complexité est quadratique (notion définie dans I)*

2. cette fonction doit agir sur place concernant le tableau `T` ie doit le modifier par effet de bord.

Question 6 *Écrire une fonction améliorée* `TriBulles2(T)` *qui sort de la fonction dès que le tableau a été parcouru sans faire d'échange (auquel cas, il est trié et donc inutile de continuer).*

Question 7 *Une variante du tri à bulles est le tri cocktail : il consiste à changer de direction à chaque passage. Lors du premier parcours, on se déplace du début du tableau vers la fin. Puis lors du second parcours on part de la fin du tableau pour arriver au début. À la troisième itération on part du début et ainsi de suite ... C'est une légère amélioration car il permet non seulement aux plus grands éléments de migrer vers la fin de la série mais également aux plus petits éléments de migrer vers le début.*

Question 8 *Écrire une fonction* `TriCocktail(T)` *qui étant donné un tableau* `T` *le renvoie trié selon la méthode du tri cocktail.*

Question 9 *Justifier en quoi cette variante peut être intéressante selon le type de tableau à trier mais vérifier néanmoins que la complexité reste quadratique.*

Utilisation Modules
Algorithmes dichotomiques
Fonctions récursives
Algorithmes gloutons
Traitement d'images
Tris