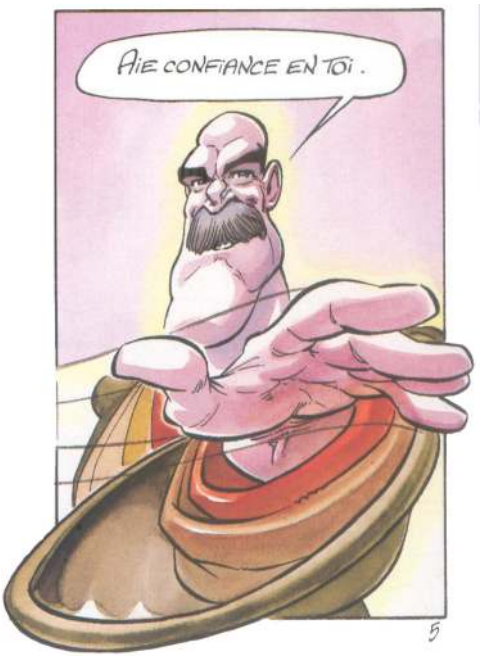
Algorithmique
Partiel n° 1 (P1)

INFO-SUP S1#
EPITA

20 Juin 2019

Consignes (à lire) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - — Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - — Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - — Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - — Aucune réponse au crayon de papier ne sera corrigée.
- ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- ☐ **Le code :**
  - — Tout code doit être écrit dans le langage `Python` (pas de C, CAML, ALGO ou autre).
  - — **Tout code `Python` non indenté ne sera pas corrigé.**
  - — Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué en **annexe** !
- ☐ Durée : 2h00



Exercice 1 (Pile ou file ? – *2 points*)

On ajoute, dans cet ordre, les valeurs A, B, C, D, E et F à une structure linéaire vide. Pour chacun des ordres de sortie donnés sur les feuilles de réponses, indiquer si la structure en question peut être : une pile, une file (ce peut être les deux), ou aucune des deux (ni une pile, ni une file).

Exercice 2 (Algorithmes de recherches - *3 points*)

Nous nous intéressons ici au *coût* d'une **recherche positive** dans une liste triée en ordre croissant (strictement).

Rappel : le *coût* d'un algorithme de recherche s'exprime en nombre de comparaisons d'éléments.

Soit une liste de n éléments qui suit une progression arithmétique de 1^{er} terme u_0 et de raison $r > 0$.

Par exemple avec $u_0 = 0$, $r = 2$ et $n = 10$, la liste sera $\lambda = \{0, 2, 4, 6, 8, 10, 12, 14, 16, 18\}$.

Remplir le tableau donné pour chacune des questions, dans les cas où $n = 20$ et $n = 100$.

— Donner une valeur pour laquelle la recherche est de coût 1. (réponse type u_5)
— Donner **une** valeur pour laquelle la recherche est de coût maximum (plusieurs réponses éventuellement possibles). Quel est ce coût ? (un entier)

Exercice 3 (Croissant – *3 points*)

Écrire la fonction `is_sorted(L)` qui vérifie si les éléments de la liste L passée en paramètre sont rangés dans l'ordre croissant.

Exemples d'applications :

```
>>> is_sorted([1, 5, 5, 12, 25])
True
>>> is_sorted([4, 3, 2, 1])
False
>>> is_sorted([4])
True
>>> is_sorted([])
True
```

Exercice 4 (Tri fusion (Merge sort) – *10 points*)

1. Écrire la fonction `partition` qui sépare une liste en deux (nouvelles) listes de longueurs quasi identiques (à 1 près) : une moitié dans chaque liste. L'ordre des éléments dans les listes résultats n'a pas d'importance.

   Exemples d'applications :

```
>>> partition([15, 2, 0, 4, 5, 8, 2, 3, 12, 25])
([15, 2, 0, 4, 5], [8, 2, 3, 12, 25])
>>> partition([5, 3, 2, 8, 7, 1, 5, 4, 0, 6, 1])
([5, 3, 2, 8, 7], [1, 5, 4, 0, 6, 1])
```

2. Écrire la fonction `merge` qui fusionne deux listes triées en ordre croissant en une seule nouvelle liste triée.

   Exemple d'application :

```
>>> merge([1, 5, 8], [2, 3, 4, 8])
[1, 2, 3, 4, 5, 8, 8]
```

3. Pour trier une liste `L`, on procède (récursivement) de la façon suivante :

   ▷ Une liste de longueur < 2 est triée.

   ▷ Une liste de longueur ≥ 2 :
   — on partitionne la liste `L` en deux sous-listes `L1` et `L2` de longueurs quasi identiques (à 1 près) ;
   — on trie récursivement les deux listes `L1` et `L2` ;
   — enfin, on fusionne les listes `L1` et `L2` en une liste triée.

   Utiliser les deux fonctions précédentes (quelles soient écrites ou non) pour écrire la fonction `sort` qui trie en ordre croissant une liste (pas en place : la fonction construit une nouvelle liste qu'elle retourne).

   Exemple d'application :

```
>>> sort([5, 3, 2, 8, 7, 1, 5, 4, 0, 6, 1])
[0, 1, 1, 2, 3, 4, 5, 5, 6, 7, 8]
```

Exercice 5 (What is it ? – *3 points*)

Soit la fonction `what` ci-dessous :

```
def what(L, x):
    i = 0
    n = len(L)
    while i < n and x > L[i]:
        i += 1
    j = i
    while j < n and x == L[j]:
        j += 1
    c = j - i
    for k in range(i, min(j, n-c)):
        L[k] = L[k+c]
    for k in range(c):
        L.pop()
```

1. Donner la valeur de la liste L après les applications de `what(`L`, `x`)` avec :
   (a) L = `[1, 2, 3, 4, 5, 6, 7]` et x = `2`
   (b) L = `[1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 5, 5, 5]` et x = `3`
   (c) L = `[1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 5, 5, 5]` et x = `5`
   (d) L = `[1, 3, 5, 7, 9]` et x = `2`
2. Soient L une liste d'entiers triée en ordre croissant et x un entier. Que fait `what(`L`, `x`)` ?

Annexe : Fonctions et méthodes autorisées

Vous pouvez utiliser les méthodes `append` , `pop` (sans argument uniquement) et la fonction `len` sur les listes :

```
>>> help(list.append)
Help on method_descriptor:    append(...)
    L.append(object) -> None -- append object to end of L

>>> help(len)
Help on built-in function len in module builtins:    len(...)
    len(object)
    Return the number of items of a sequence or collection.

>>> help(list.pop)  # cannot be use with index here...
Help on method_descriptor:
pop(...)
    L.pop() -> item -- remove and return last item.
    Raises IndexError if list is empty.
```

Aucun opérateur n'est autorisé sur les listes (`+`, `*`, `==` ...).

Vous pouvez également utiliser la fonction `range` et `raise` pour déclencher les exceptions. Rappels :

```
>>> for i in range(10):
...     print(i, end=' ')
0 1 2 3 4 5 6 7 8 9

>>> for i in range(5, 10):
...     print(i, end=' ')
5 6 7 8 9

>>> raise Exception("blabla")
...
Exception: blabla
```

Vos fonctions

Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).

Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.