FEUILLE n°5 *Algorithique et complexité*

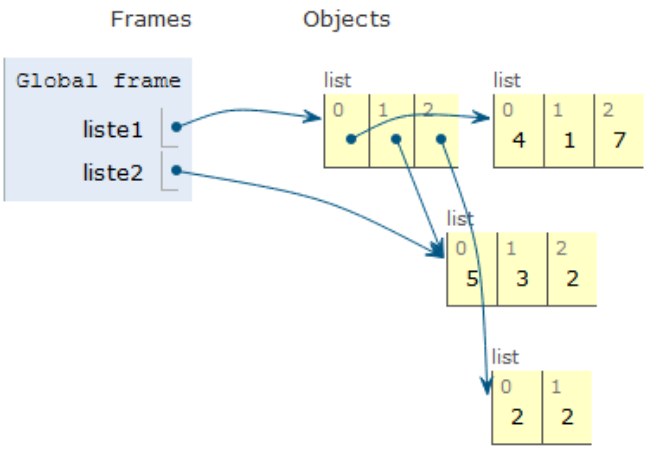
## Exercice 1. Question de cours

```
pokedex_anakin_v1 = {
    ('Carmache', 'Dragon'), ('Carmache', 'Sol'),
    ('Colimucus', 'Dragon'), ('Palkia', 'Dragon'),
    ('Palkia', 'Eau')}

pokedex_anakin_v2 = {
    'Carmache': {'Dragon','Sol'},
    'Colimucus': {'Dragon'},
    'Palkia': {'Dragon', 'Eau'}}

pokedex_anakin_v3 = {
    'Dragon': {'Carmache','Colimucus', 'Palkia'},
    'Sol': {'Carmache'},
    'Eau': {'Palkia'}}

a = max(pokedex_anakin_v1)
b= min(pokedex_anakin_v2)
c = sorted(pokedex_anakin_v3)
```

**1.1.** préciser le type et la valeur des variables `a`, `b` et `c` l'issue du script.

## Exercice 2. Représentation de la mémoire

**2.1.** Proposer un script qui provoque la représentation de la mémoire ci-contre.




**2.2.** préciser la valeur de `liste1` et `liste2` si j'ajoute l'instruction suivante :

```
liste2.append(5)
```

**2.3.** préciser la valeur de `liste1` et `liste2` si j'ajoute l'instruction suivante :

```
liste1.append(5)
```

### Exercice 3. Lecture de code et complexité

**3.1.** Compléter les tests dans le script ci-dessous

**3.2.** Déterminer la complexité de chacune des fonctions

**3.3.** Associer chaque fonction à sa description

```
mes_nombres = [17, 14, 10, 17, 19, 10, 17, 17, 14]

def fonction1(liste): # Complexité = ??
    res = liste[0]
    for nombre in liste:
        if res > nombre:
            res = nombre
    return res
assert fonction1(mes_nombres) == ???

def fonction2(liste): # Complexité = ??
    res = sorted(liste)
    return res[0]
assert fonction2(mes_nombres) == ???

def fonction3(liste): # Complexité = ??
    return max(liste)
assert fonction3(mes_nombres) == ???

def fonction4(liste): # Complexité = ??
    for i in range(len(liste)):
        for j in range(len(liste)):
            if liste[i] == liste[j] + 1:
                return True
    return False
assert fonction4(mes_nombres) == ???

def fonction5(liste): # Complexité = ??
    ensemble = set(liste)
    return len(liste) == len(ensemble)
assert fonction5(mes_nombres) == ???

def fonction6(liste): # Complexité = ??
    ensemble = set(liste)
    return len(ensemble)
assert fonction6(mes_nombres) == ???

def fonction7(liste): # Complexité = ??
    aux = sorted(liste)
    for i in range(len(aux)-1):
        if aux[i+1] == aux[i] + 1:
            return True
    return False
assert fonction7(mes_nombres) == ???
```

```
""" doc A
renvoie le nombre
d'éléments de la liste
"""

""" doc B
renvoie le plus petit
nombre de la liste
"""

""" doc C
vérifie si la liste
contient au moins deux
nombres consécutifs
"""

""" doc D
renvoie la liste triée
dans l'ordre croissant
"""

""" doc E
renvoie le plus grand
nombre de la liste
"""

""" doc F
compte le nombre d'éléments
distincts de la liste
"""

""" doc G
vérifie si la liste
contient au moins deux
nombres identiques
"""
```

### Exercice 4. Quelques fonctions sur des listes triées

**4.1.** Le slice des séquences est un outil pratique en python. Compléter les exemples suivants :

```
liste = ['a', 'b', 'c' 'd', 'e', 'f', 'g', 'h']}
liste1 = liste[1:5]  # liste1 = ['b', 'c', 'd' 'e']
liste2 = liste[6:]   # liste2 = ['g', 'h']
liste3 = liste[:3]   # liste3 = ['a', 'b', 'c']
liste5 = liste[2:4]  # liste5 = ???
liste6 = liste[???]  # liste6 = ['a', 'b']
liste7 = liste[5:11] # liste7 = ???
```

**4.2.** Compléter le code de la fonction `mystere` (documentation et tests) et préciser sa complexité.

```
def mystere (liste1, liste2):
    """ paramètres: liste1 , liste2 deux listes triées dans l'ordre croissant
        résultat : ??? """
    res = []
    (i1, i2) = (0, 0)
    while i1 < len(liste1) and i2 < len(liste2):
        (e1, e2) = (liste1[i1] , liste2[i2])
        if e1 <= e2:
            res.append(e1)
            i1+= 1
        else:
            res.append(e2)
            i2+= 1
    res = res + liste1[i1:] +liste2[i2:]
    return res

assert mystere([1, 1, 2, 5, 6, 7, 8, 9], [3, 9, 10, 10, 12]) == ??
```

**4.3.** Compléter le code de la fonction `intersection`.

```
def intersection(liste1, liste2):
    """
    Paramètres: liste1 , liste2 deux listes triées dans l'ordre croissant
    Résultat : la liste des éléments présents dans les deux listes, le nombre
    d'occurrence de chaque élément étant égal au minimum de son nombre
    d'occurrence dans liste1 et liste2
    """
    pass

assert intersection([1, 1, 2, 3], [4, 5, 6]) == []
assert intersection([1, 1, 2, 3, 6, 7], [3, 4, 5, 6, 8]) == ???
assert intersection([1, 1, 2, 3], ???) == [2, 3]
assert intersection([1, 1, 2, 3], [1, 1, 1, 1]) == [1, 1]
assert intersection([1, 1, 2, 2, 3], [1, 1, 1, 2]) == [1, 1, 2]
assert intersection([1, 1, 2, 3], [???]) == [1, 2, 3]
```

Récupérer le fichier `feuille5.py` sur Célène

### Exercice 5. L'Association des Dresseurs de Pokemons d'Orléans

L'ADPO (Association des Dresseurs de Pokemons d'Orléans) modélise l'ensemble de ses dresseurs par un dictionnaire dont les clefs sont les noms des dresseurs et la valeur associée son nombre de points au classement Elo. Par exemple :

```
novembre_2020 = {'Jasmine': 1400, 'Will': 1610, 'Zoé': 1100, 'Sasha': 2005,
    'Théo': 700, 'Morgan': 1700, 'Maxime': 1650, 'Florent': 1800, 'Olive': 1500}
decembre_2020 = {'Jasmine': 1510, 'Zoé': 980, 'Morgan': 1650, 'Sasha': 1880,
    'Olive': 1670, 'Florent': 1810, 'Will': 1460, 'Théo': 850, 'Maxime': 1400}
```

L'ADPO organise tous les mois le *Duel du mois*, un spectacle où s'affrontent deux dresseurs de l'association. Pour que le match soit le plus intéressant possible, les deux dresseurs choisis sont les dresseurs qui ont les classements Elo les plus proches. Par exemple, au mois de novembre 2020, le *Duel du mois* a opposé Will et Maxime Au mois de décembre 2020, le *Duel du mois* a opposé Morgan et Olive

**5.1.** Compléter le code de la fonction `duellistes` qui renvoie le nom des duellistes du mois. Vous pouvez écrire des fonctions auxilliaires.

```
def duellistes(dresseurs):
    """
    param: dresseurs est un dictionnaires :
    - clé: le nom du dressur (str)
    - valeur : son classement Elo (int)
    résultat: renvoie le nom des deux dresseurs qui ont les classements
    Elo les plus proches
    """
    pass

assert duellistes(novembre_2020) == ('Will', 'Maxime')
assert duellistes(decembre_2020) == ('Morgan', 'Olive')
```

**5.2.** Quelle est la complexité de votre fonction ?

### Exercice 6. Quelques fonctions sur des listes (triées ou non)

Quand on travaille avec des listes triées, un grand nombre de fonctions sont plus simples à écrire et plus efficaces que sur des listes quelconques.

**6.1.** La fonction suivante garde la première occurrence de chaque élément d'une liste. Quelle est sa complexité ?

```
def rend_unique(liste):
    """
    parametre: une liste d'éléments comparables
    résultat: une liste contenant les éléments de 'liste', une fois chacune
    de leur première occurence .
    """
    deja_vus = set ()
    res = []
    for elem in liste:
        if elem not in deja_vus :
            res.append(elem)
            deja_vus.add(elem)
    return res

assert rend_unique([1, 6, 5, 1, 4, 7, 1, 2, 5, 6, 1, 7]) == [1, 6, 5, 4, 7, 2]
```

**6.2.** Compléter le code de la fonction `rend_unique_liste_triee` qui prend en paramètre une liste que l'on suppose triée dans l'ordre croissant et renvoie une copie de cette liste, sans répétitions. Préciser sa complexité.

```
def rend_unique_liste_triee(liste):
    """
    parametre: une liste dont les éléments sont triés dans l'ordre croissant
    résultat: une liste contenant les éléments de 'liste', une fois chacune
    de leur première occurence .
    """
    pass

assert rend_unique([1, 1, 2, 5, 5, 5, 5, 7, 8, 9, 9]) == [1, 2, 5, 7, 8, 9]
```

**6.3.** **En bonus** Écrire une fonction `plus_grande_plage_consecutive` qui prend en entrée une liste d'entiers et renvoie la plus grande liste d'entiers contenus dans cette liste dont tous les éléments se suivent : par exemple, `consecutifs([5, 7, 6, 13, 15, 14, 12])` renvoie [12, 13, 14, 15]

### Exercice 7. Lecture de code

```
def recherche(liste, cible):
    """
    param:
      - liste est une liste de nombres triés dans l'ordre croissant
      - cible est un nombre
    résultat : ???
    """
    bas = 0
    haut = len(liste) -1
    while bas <= haut:
        milieu = (haut + bas) // 2
        if cible < liste[milieu]:
            haut = milieu -1
        else:
            bas = milieu + 1
    return (liste[haut] == cible)

exemple = [3, 6, 9, 11, 12, 15, 18, 21]
assert recherche(exemple, 13) == ???
assert recherche(exemple, 14) == ???
assert recherche(exemple, 15) == ???
```

**7.1.** Compléter le tableau suivant en précisant la valeur des différentes expressions à chaque tour de boucle lors de l'appel : `recherche([0, 3, 6, 9, 12, 15, 18, 21], 13)`

| | `haut` | `bas` | `milieu` | `liste[milieu]` | `cible < liste[milieu]` |
|---|---|---|---|---|---|
| avant la boucle | | | | | |
| pendant la boucle | | | | | |
| à la sortie de la boucle | | | | | |

**7.2.** Compléter le code de la fonction (documentation et tests).

**7.3.** Compléter le tableau suivant, en indiquant notamment la valeur des variables `haut` et `bas` *à la sortie de la boucle* pour chaque entrée.

| `liste` | `cible` | résultat | valeur de `haut` à la sortie de la boucle | valeur de `bas` à la sortie de la boucle |
|---|---|---|---|---|
| `[0, 3, 6, 9, 12, 15, 18, 21]` | 13 | | | |
| `[0, 3, 6, 9, 12, 15, 18, 21]` | 14 | | | |
| `[0, 3, 6, 9, 12, 15, 18, 21]` | 15 | | | |
| `[0, 3, 6, 9, 12, 15, 18, 21]` | 25 | | | |

**7.4.** Pourquoi est-on sûr de sortir de la boucle while au bout d'un moment ?

**7.5.** Quel est la complexité de la fonction `recherche` ?

### Exercice 8. (Facultatif) Un peu d'algorithmique

On cherche à écrire une fonction qui, à partir d'une liste de nombres et d'un nombre «cible», détermine le nombre de la liste qui est le plus proche de la cible.

#### Dans le cas général

Avec la liste `[17, 2, 15, 13, 28, 4, 11]` et la cible `7`, le résultat attendu sera 4 car c'est le nombre de la liste qui est le plus proche de 7.

**8.1.** Compléter le tableau suivant :

| liste | cible | résultat attendu |
|---:|:---:|:---:|
| `[1, 2, 5, 3]` | 2 | |
| `[11, 1, 2, 55, -1, 5, 3]` | 0 | |
| `range(-5,100,4)` | 95 | |

**8.2.** Donner le code d'une fonction qui résout ce problème. Préciser sa complexité.

#### On suppose maintenant que la liste de nombres est triée

**8.3.** Quel type d'algorithme peut-on utiliser pour trouver un nombre dans une liste triée avec une meilleure complexité que la fonction de la question précédente ?

**8.4.** Donner le code d'une fonction qui résout le problème avec une liste triée. Préciser sa complexité.

### Exercice 9. (Facultatif) Locations de surf

La boutique *Silver Surfing Coconut Club* du Charlemagne-Island Beach Resort propose de louer des planches de surf pour la journée. Nous allons aider le gérant à mettre en place un système de réservation en ligne pour les planches.

Le Silver Surfing Coconut Club dispose d'un certain nombre de planches de tailles différentes. Ces planches doivent être attribuées en début de journée aux personnes qui ont fait une réservation. Ces personnes ont indiqué leur taille au moment de leur réservation.

Les planches disponibles à la boutique sont représentées par une liste *triée* d'entiers. Ces entiers représentent la taille de chaque planche en centimètres (on considère qu'à part la taille, les planches sont identiques). Voici un exemple d'une telle liste :

```
planches_exemple = [152, 161, 161, 170, 175, 185, 190, 200]
```

**9.1.** Les personnes qui viennent louer des planches donnent leur nom et leur taille. Par exemple, voici les personnes qui viennent louer des planches le lundi :

| | | | |
|---|---|---|---|
| Alex (182 cm) | Bachir (172 cm) | Chen (171 cm) | Dalila (173 cm) |
| Estéban (179 cm) | Fanta (165 cm) | Gérard (195 cm) | Henrietta (156 cm) |

Et voici les personnes qui viennent louer des planches le mardi :

| | | | |
|---|---|---|---|
| Alex (182 cm) | Bachir (172 cm) | Chen (171 cm) | Dalila (173 cm) |
| Estéban (179 cm) | Fanta (165 cm) | | |

Par quelle structure de données peut-on les représenter ?

```
personnes_exemple_lundi = ???
personnes_exemple_mardi = ???
```

Pour une meilleure pratique du surf, il est préférable que la taille de la planche soit la plus proche possible de la taille de chaque personne [1]. On cherche donc à attribuer les planches de sorte que chaque personne ait une planche dont la taille soit la plus proche possible de la sienne.

**9.2.** Écrire une fonction `attribution(planches, personnes)` qui, à partir d'une liste de taille de planches et de les personnes qui se sont inscrites, renvoie un dictionnaire où chaque clé est le nom d'une personne et la valeur associée est la taille de la planche qui lui est attribuée.

```
assert attribution(planches_exemple, personnes_exemple_lundi) == {
    'Alex': 190, 'Bachir': 170, 'Chen': 161, 'Dalila': 175,
    'Estéban': 185, 'Fanta': 161, 'Gérard': 200, 'Henrietta' : 152}
assert attribution(planches_exemple, personnes_exemple_mardi) == {
    'Alex': 200, 'Bachir': 175, 'Chen': 170, 'Dalila': 185,
    'Estéban' : 190, 'Fanta':161}
```

1. Avertissement : La personne qui a rédigé cet énoncé enseigne les structures de données, pas le surf. Son expérience du surf se résume à l'écoute de *Pet Sounds* des Beach Boys. La pratique du surf est interdite sur le lac de l'université.