Retour sur les tris

Un peu d'histoire

Plankalkül est un langage de programmation, conçu de 1942 à 1946 par l'Allemand Konrad Zuse. D'après lui, il est le premier langage de haut niveau. Ce langage présente une innovation rare, voire unique, pour un langage informatique : les programmes s'écrivent en deux dimensions, de la même façon que la notation algébrique. Il est donc plus proche des traditions du plan et de la formule que de celles de la prose et du récit.

Cours

En python, l'opérateur de comparaison < (et ses homologues <=, > et >=) fonctionne différemment suivant les types de variables que l'on compare. Par exemple :

- il compare les nombres de façon naturelle ;

Expression	Valeur
`8 < 13`	`True`
`6 < -1`	`False`

- il compare les caractères suivant leur position dans la table ASCII ;

Expression	Valeur
`'D' < 'V'`	`True`
`'u' < 'j'`	`False`
`'a' < 'A'`	`False`

- il compare les chaînes caractère par caractère en considérant que `''` est le minimum :

Expression	Valeur
`"DAVID" < "DAVID W"`	`True`
`"Judo" < "Judas"`	`False`
`"Nsi" < "nsI"`	`True`

- il compare les tuples élément par élément en considérant que le tuple vide est le minimum :

Expression	Valeur
`(12, 8, 5) < (13, 1, 7)`	`True`
`(12, 8, 5) < (12, 1, 7)`	`False`
`(12, 8) < (12, 8, 4)`	`True`

et il agît de la même façon pour les listes.

Il est par contre impossible d'utiliser l'opérateur < entre deux dictionnaires.

D.W.

Automatismes

Donner la valeur de :

- ❏ `18-4**2 < 4`
- ❏ `[3, 1, 4, 0] < [3, 1, 4]`

Donner la valeur de :

- ❏ `"Info" < "Intro"`
- ❏ `(25, 7, 1989) < (23, 7, 1990)`

Exercice 1 : Comparer des dates

On suppose qu'une date est modélisée par un tuple de la forme `(jj, mm, aaaa)`.
Par exemple, le 14 juillet 1789 est représenté par le tuple `(14, 7, 1789)`.

Une **fonction de comparaison** entre deux dates est une fonction `cmp_date` à deux arguments `t1` et `t2` qui vérifie :

$$\texttt{cmp\_date(t1, t2)} = \begin{cases} 1 & \text{si t1 est antérieur à t2;} \\ 0 & \text{si t1 est égal à t2;} \\ -1 & \text{si t1 est postérieur à t2.} \end{cases}$$

1. Que renvoie l'instruction `cmp_date((22, 7, 1987), (25, 7, 1989))` ?

2. Compléter le code ci-dessous de la fonction `cmp_date` :

```
def cmp_date(t1:tuple, t2:tuple) -> int:
    """ On suppose que les dates sont valides. """
    j1, m1, a1 = t1
    j2, m2, a2 = t2

    if ... < ... :
        res = 1
    elif ... > ... :
        res = -1
    else:
        if ... < ... :
            res = 1
        elif ... > ... :
            res = -1
        else:
            if ... < ... :
                res = 1
            elif ... > ... :
                res = -1
            else:
                res = 0
    return res
```

3. Réécrire cette fonction en comparant directement des tuples.

Exercice 2 : Comparer des élèves

On considère qu'un élève peut être représenté par trois attributs donnés par :

❏ un nom ❏ un prénom ❏ une date de naissance

On décide d'utiliser des **tuples nommés** pour représenter des élèves.

Cours

Un **tuple nommé** se comporte comme un tuple où l'accès aux éléments indexés peut se faire à l'aide de nom d'attributs. Il faut importer ce type d'objet à l'aide de l'instruction :

```
from collections import namedtuple
```

Par exemple, pour créer des objets de type **élève**, on écrit :

```
Eleve = namedtuple("ELEVE", ["nom", "prenom", "naissance"])
```

et il est alors possible de créer des instances d'`Eleve` avec des instructions comme :

```
el1 = Eleve("Wrobel", "David", (22, 7, 1987))
```

Les instructions `el1.nom`, `el1.prenom` et `el1.naissance` renvoient alors successivement `"Wrobel"`, `"David"` et `(22, 7, 1987)`.

1. Quelles instructions permettent de créer les élèves suivants :

Variable	Nom	Prénom	Naissance
el1	"Robert"	"Lucas"	(13, 1, 2007)
el2	"Lemoine"	"Sophie"	(26, 6, 2008)

2. Que renvoie l'instruction `cmp_date(el1.naissance, el2.naissance)` ?

3. On dirait qu'un élève est inférieur à un autre s'il est plus jeune.
Écrire une fonction `cmp_eleve(el1:namedtuple, el2:namedtuple) -> int` qui compare deux élèves.

Exercice 3 : Comparer des élèves suivant la moyenne

On décide de modéliser un élève par un dictionnaire dont les entrées sont :

❏ un nom ❏ un prénom ❏ une date de naissance ❏ un ensemble de notes

Par exemple, on pourrait avoir :

```
el1 = { "nom"       : "Wrobel",
        "prenom"    : "David",
        "naissance" : (22, 7, 1987),
        "notes"     : [13, 18, 4, 16]}
```

1. Écrire la fonction `ajouter_note(el:dict, val:int) -> None` qui ajoute une note à un élève donné.

2. Écrire la fonction `moyenne(el) -> float` qui calcule la moyenne d'un élève donné.
On n'utilisera ni la fonction `sum`, ni la fonction `len`.

3. Compléter la fonction `cmp_eleve_moyenne(el1:dict, el2:dict) -> int` qui compare les élèves passés en paramètre suivant leur moyenne.

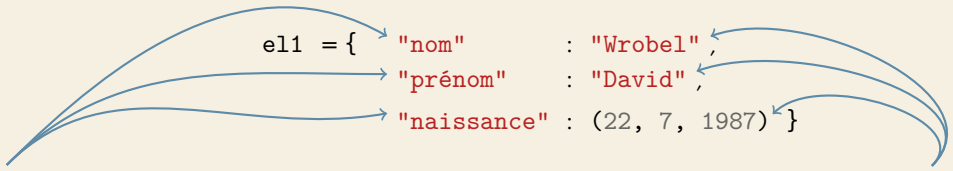
```
def cmp_eleve_moyenne(el1:dict, el2:dict) -> int:
    """ Comparaison d'élèves. """
    moy1, moy2 = ..., ...

    if ... < ... :
        res = 1
    elif ... > ... :
        res = -1
    else:
        res = 0

    return res
```

Cours

En python, **Un dictionnaire** est une collection d'objets accessibles non pas via un index mais via une clé.
Pour définir un dictionnaire, on utilise une paire d'accolades { } :



Les clés sont forcément des objets immuables. Il n'y a pas de restriction sur les valeurs

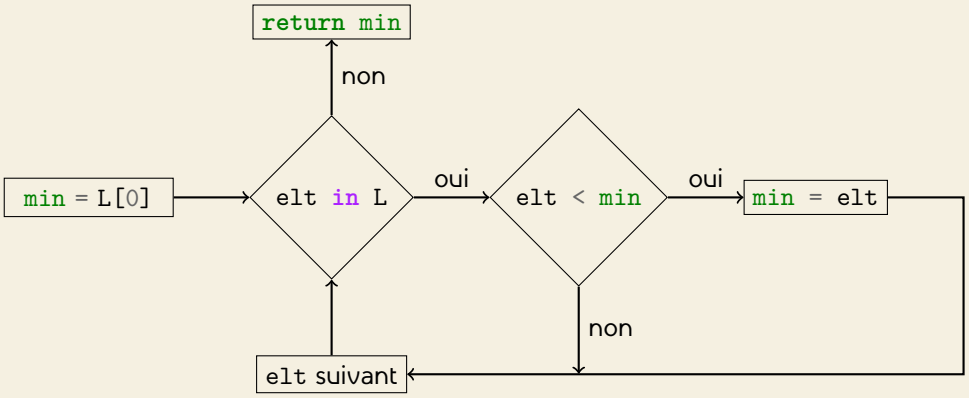
Pour accéder aux valeurs d'un dictionnaire, on utilise la notation entre crochets [] :

`el1["nom"]` renvoie `"Wrobel"`

Pour ajouter un élément à un dictionnaire, on lui crée une entrée avec `el1["notes"] = [12, 5, 3, 18]`.

Exercice 4 : Recherche du minimum

On s'intéresse dans cet exercice à la recherche de l'élément minimum dans une liste d'entiers.
Pour cela, on s'intéresse à l'algorigramme suivant :



1. Faire tourner cet algorigramme sur la liste `L = [13, 15, 24, 8, 12, 7, 14]`.
On complétera le tableau de variables ci-dessous :

i	L[i]	min
1	15	13

2. Proposer une fonction `recherche_min(L:list) -> int` qui recherche le minimum d'une liste.
On utilisera une boucle bornée sur les éléments.

Exercice 5 : Indice du minimum

On s'intéresse à une liste de dates (de la forme `(jj, mm, aaaa)`) et on cherche à obtenir l'indice de la date la plus ancienne. On suppose que la fonction `cmp_date(t1:tuple, t2:tuple) -> int` est toujours définie.

On propose la fonction suivante :

```
def recherche_indice_min(L:list) -> int:
    """ Recherche l'indice minimum d'une liste non vide de dates. """
    res = 0
    for i in range(1, len(L)):
        if L[i] < L[res]:
            res = i
    return res
```

1. Que renvoie cette fonction pour la liste ci-dessous ?

```
L = [(14, 1, 1984), (22, 7, 1987), (19, 8, 1985), (6, 1, 1961), (17, 1, 1964)]
```

2. Proposer une correction pour que cette fonction renvoie réellement l'indice de la date la plus ancienne.

Cours

Pour trier une liste de `n` éléments à l'aide du **tri par sélection**, on effectue les opérations suivantes :

- ❏ On échange l'élément d'indice 0 avec le plus petit élément de la liste ;
- ❏ On échange l'élément d'indice 1 avec le deuxième plus petit élément de la liste ;
- ❏ On échange l'élément d'indice 2 avec le troisième plus petit élément de la liste ;
- ❏ etc.

Par exemple, pour la liste `L = [13, 5, 18, 10, 2, 9, 16]` on aurait :

La liste `L`	Commentaire
`[13, 5, 18, 10, 2, 9, 16]`	Liste de départ

Pour trier cette liste, on a utilisé un total de ☐ comparaisons.

De façon générale, pour trier une liste de taille `n`, on compare le `i`-ième élément avec ☐ autres.

Cela représente un total de ☐ comparaisons.

Exercice 6 : Implantation du tri par sélection

1. On considère la liste `L = [12, 64, 32, 15, 19, 27, 8, 16]`.

a. Donner l'état de la liste L après chaque passage de boucle du tri par sélection.

b. Combien de comparaisons ont été utilisées pour trier cette liste ?

On se propose d'implanter le code du tri par sélection pour une liste d'entiers.
Pour cela, on propose le pseudo-code suivant :

```
fonction tri_selection(liste L)
    n = longueur(L)
    pour i de 0 à n - 2
        mini = i
        pour j de i + 1 à n - 1
            si t[j] < t[mini], alors mini = j
        fin pour
        si mini != i, alors échanger t[i] et t[mini]
    fin pour
```

2. a. Expliquer pourquoi la première boucle varie de `0` à `n-2`.

b. i. Pourquoi initialise-t-on la variable `min` à `i` ?

ii. Pourquoi l'indice de la seconde boucle varie de `i+1` à `n-1` ?

c. À quoi sert la dernière condition ?

3. Proposer un code python pour la fonction `tri_selection(L:list) -> None`.

4. Que renvoie cette fonction pour `L = ['Ph', 'NSI', 'Maths', 'SVT', 'SI']` ?

Cours

La fonction permettant d'effectuer un **tri par sélection** d'une liste L est donnée par :

Exercice 7 : Variante du tri par sélection

Une variante du tri par sélection consiste non pas à chercher l'indice du minimum mais plutôt l'indice du maximum. On échange alors ce maximum avec le dernier élément de la liste et on poursuit la recherche. La liste est alors triée de droite à gauche.

1. Donner les états successifs de la liste `L = [14, 4, 18, 19, 7, 10, 5, 16]` avec cette variante.

2. Compléter le code ci-dessous correspondant à cette variante :

```
def variante_tri_selection(L:list) -> None:
    """   Tri de droite à gauche. """
    n = len(L)

    for i in range(0, n-1) :
        maxi = ...

        for j in range(0, n-i):
            if L[j] ... L[maxi]:
                maxi = j

        if ... != maxi:
            L[...], L[maxi] = L[maxi], L[...]

```

3. Réaliser cette variante avec la liste `L = ["David", "Laurie", "Bastien", "Alexis"]`.

Cours

Pour trier une liste de `n` éléments à l'aide du **tri par insertion**, on effectue les opérations suivantes :

- ❏ On décale l'élément d'indice 1 vers la gauche tant qu'il n'est pas à la bonne place ;
- ❏ On décale l'élément d'indice 2 vers la gauche tant qu'il n'est pas à la bonne place ;
- ❏ On décale l'élément d'indice 3 vers la gauche tant qu'il n'est pas à la bonne place ;
- ❏ etc.

Par exemple, pour la liste `L = [13, 5, 18, 10, 2, 9, 16]` on aurait :

La liste `L`	Commentaire
`[13, 5, 18, 10, 2, 9, 16]`	Liste de départ

Remarque : On ne peut pas décaler l'élément d'indice 0.

Cours

Pour trier cette liste, on a utilisé un total de ☐ comparaisons.

De façon générale, pour trier une liste de taille `n`, on compare le `i`-ième élément avec :

- ❑ au maximum ☐ autres ;
- ❑ au minimum ☐ autre.

Le nombre total de comparaisons peut ainsi être encadré par :

Exercice 8 : Application

On considère la liste `L = [12, 64, 32, 15, 19, 27, 8, 16]`.

1. Donner l'état successif de la liste `L` après chaque étape du tri par insertion.
2. Combien de comparaisons ont été effectuées ?

Exercice 9 : Meilleur et pire des cas

On considère les listes `L1 = [1, 2, 3, 4, 5]` et `L2 = [5, 4, 3, 2, 1]`.

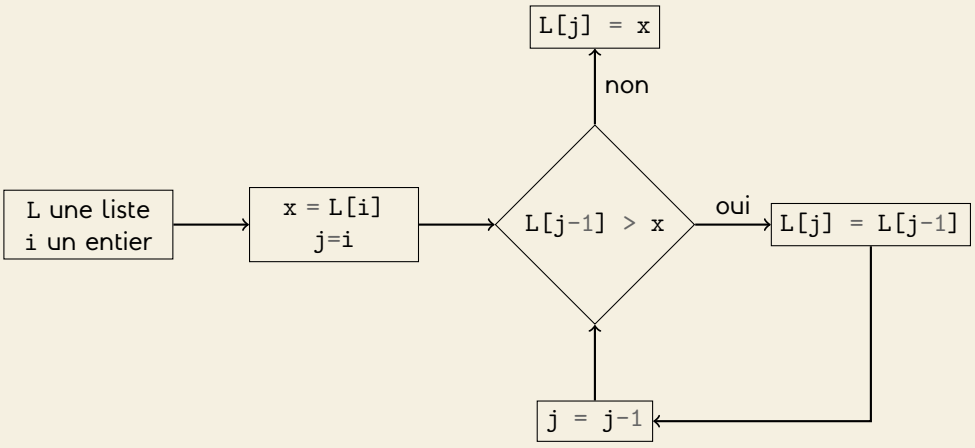
1. Donner le nombre de comparaisons lorsqu'un tri par insertion est effectué sur chacune de ces listes.

Exercice 10 : Décalage à gauche

On souhaite écrire une fonction `decalage(L:list, i:int) -> None` qui décale vers la gauche l'élément d'indice i de sorte que

$$L[i-1] \leqslant L[i] \leqslant L[i+1]$$

On propose l'algorigramme suivant :



On considère la liste `L = [7, 12, 16, 10, 4]`.

1. Donner les états successifs de la liste `L` lors de l'instruction `decalage(L, 3)`.
2. **a.** Quelle erreur produit l'instruction `decalage(L, 4)` ?

   b. Proposer une correction du bloc conditionnel.

   c. Écrire le code python associé à cet algorigramme.

Cours

La fonction permettant d'effectuer un **tri par insertion** d'une liste `L` est donnée par :

Exercice 11 : Tri par insertion de chaînes

Dans cet exercice, on dira qu'une chaîne `c1` est inférieure à une autre `c2` si `len(c1) < len(c2)`.

1. Donner les états successifs de la liste

```
L = ["Maths", "Physique", "SVT", "SI", "NSI"]
```

lors d'un tri par insertion.

2. Proposer une fonction `cmp_string(c1:str, c2:str) -> int` qui compare deux chaînes de caractères.

3. Compléter le code de la fonction `tri_insertion(L:list) -> None` ci-dessous pour qu'elle puisse trier une liste de chaînes de caractères :

```
def tri_insertion(L:list) -> None:
    """ Trie par insertion une liste de chaînes de caractères. """

    for i in range(1, len(L)):

        x = L[i]
        j = i

        while j > 0 and ... :

            L[j] = L[j-1]
            j = j-1

        L[j] = x
```