Chapitre (ANN) 3 Synthèse pour l'algorithmique

1 Généralités .....
2 Scripts généraux .....
3 Scripts sur les tris .....

Résumé & Plan

Un condensé de commandes importantes et de scripts classiques à connaitre parfaitement pour les concours.

1. GÉNÉRALITÉS

1.1. Opérations élémentaires

■■ Arithmétique

```
>>> 5**2 # puissance
25
>>> 5//2 # quotient division
2
>>> 5%2 # reste division
1
```

(Gestion de variables) `x += 2` : ajoute 2 à la variable x
`x *= 2` : multiplie x par 2
`x, y = y, x` : échange x et y

Les échanges fonctionnent également dans une liste, par exemple :

```
>>> L = [1, 2, 3]
>>> L[0], L[2] = L[2], L[0]
>>> L
[3, 2, 1]
```

1.2. Types

On rappelle que les types `tuple`, `str` par exemples sont immuables (impossible donc de modifier leurs éléments en place).

(Types & Conversions) `type(x)` : type des de la variable x
`int` (entier), `float` (réel)
`bool` (booléen : `True` ou `False`)
`list` (liste), `dict` (dictionnaire), `tuple` (tuple), `str` (chaîne de caractères), `array` (tableau).
Exemples de conversions de types.

```
>>> int(2.5)
2
>>> float(2)
2.0
>>> str(2)
'2'
>>> int("3")
3
```

(Affichage et récupération de variables) `x = int(input("Valeur de x"))` : demander une valeur
`a, b, c = 1, 2.5, "blabla"` : affectation multiple
`print("la valeur de x est", x)` : afficher, `del(x)` : effacer la variable x,

MANIPULATIONS DE LISTES

Commande	Effet
`len(L)`	longueur
`L[0]`	premier élément
`L[-1]`	dernier élément
`L[i:j]`	liste extraite des éléments d'indices entre i (inclus) et j (exclus)
`L[i:]`	liste extraite à partir de l'indice `i` (inclus)
`L[i:]`	liste extraite jusqu'à l'indice `j` (exclu)
`L.append(v)`	ajoute l'élément `v` à la fin de la liste
`L.remove(v)`	supprimer le premier élément `v` apparaissant dans la liste,
	renvoie une erreur s'il n'est pas présent
`L.extend(s)`	ajoute la liste `s` à la fin de la liste
`L.insert(i, v)`	insert l'objet `v` à l'indice `i`
`del L[i]`	supprime l'élément d'indice `i`
`del L[i:j]`	supprime les éléments entre les indices `i` et `j` si `j > i`
`L.pop()`	supprime le dernier élément et renvoie l'élément supprimé

MANIPULATIONS DE CHAÎNES

Commande	Effet
`len(s)`	longueur
`s[0]`	premier élément
`s[-1]`	dernier élément
`s[i:j]`	chaîne extraite des éléments d'indices entre i (inclus) et j (exclus)
`s[i:]`	chaîne extraite à partir de l'indice i (inclus)
`s.join`	concatène à la chaîne de départ la nouvelle
`L = s.split()`	affecte à `L` la chaîne `s` dont les mots ont été séparés (et toute la chaîne si elle n'a pas de trou).

1.3. Tests

Voici un bref panorama des test.

Tests : `==, !=, <=, >=, <, >`
Opérateurs logiques : **`and, or, not`**

Version courte
```
if test:
    instructions
    ...
```

Avec alternative
```
if test:
    instructions 1
    ...
else:
    instructions 2
    ...
```

Avec plusieurs tests
```
if test1:
    instructions 1
    ...
elif test2:
    instructions 2
    ...
else :
    instructions 3
    ...
```

1.4. Boucles

(Générateurs sur des entiers/Listes d'entiers)
`range(n)` : entiers entre 0 et $n-1$
`range(a, b)` : entiers entre a et $b-1$
`range(a, b, k)` : entiers entre a et $b-1$ en avancent avec un pas de k

Attention
On rappelle que les objets précédents ne sont pas du type `list`. Pour les convertir en liste, on utilise `list(range(a,b))` par exemple.

Boucle for
```
for k in sequence:
    instructions
```

Boucle while
```
while test:
    instructions
```

2. SCRIPTS GÉNÉRAUX

2.1. Sur les entiers

(Factorielle)

```
def factorielle(n):
    P = 1
    for k in range(1, n+1):
        P *= k
    return P
```

```
def factorielle_rec(n):
    if n == 0:
        return 1
    else:
        return \
        ↪ n*factorielle_rec(n-1)
```

2.2. Sur les listes

(Appartenance d'un élément dans une liste par balayage)

```
def appartient(x, L):
    """
    renvoie True si x est dans L
    """
    for y in L:
        if y == x:
            return True
    return False
```

(Calcul d'une somme)

```
def somme(L):
    """
    renvoie la somme des éléments de L
    """
    S = 0
    for x in L:
        S += x
    return S
```

(Calcul du produit)

```
def produit(L):
    """
    renvoie le produit des éléments d'une liste
    """
    P = 1
    for x in L:
        P *= x
    return P
```

(Calcul du maximum)

```
def maximum(L):
    """
    renvoie la valeur du maximum de L
    """
    maxi = L[0]
    for x in L[1:]:
        if x > maxi:
            maxi = x
    return maxi
```

(Calcul du maximum + positions où il est réalisé)

```
def maximum_occur(L):
    """
    renvoie le maximum de L, et renvoie la liste des occurences \
    ↪ où
    il apparaît
    """
    # Recherche du maximum
    maxi = maximum(L)
    # Recherche des indices
    L_ind_maxi = []
    for k in range(len(L)):
        if L[k] == maxi:
            L_ind_maxi.append(k)
    return maxi, L_ind_maxi

def maximum_occur_bis(L):
    """
    renvoie le maximum de L, et renvoie la liste des occurences \
    ↪ où
```

```
    il apparaît.
    Un seul parcours de la liste ici.
    """
    maxi = L[0]
    L_ind_maxi = [0]
    for k in range(1, len(L)):
        if L[k] == maxi:
            L_ind_maxi.append(k)
        if L[k] > maxi:
            maxi = L[k]
            # découverte d'un nouveau potentiel max, on vide la liste
            L_ind_maxi = [k]
    return maxi, L_ind_maxi
```

(Calcul du maximum + position du premier indice)

```
def maximum_premind(L):
    """
    renvoie le maximum de L, et renvoie le premier indice où il apparaît
    """
    maxi = L[0]
    ind_maxi = 0
    for k in range(1, len(L)):
        if L[k] > maxi:
            maxi = L[k]
            ind_maxi = k
    return maxi, ind_maxi
```

Il faut savoir également adapter cette fonction au dernier indice, ainsi que toutes les autres au minimum.

2.3. Sur les chaînes de caractère

(Recherche d'un mot dans une chaîne)

```
def cherche_mot(mot, ch):
    """
    Recherche le mot m dans une chaîne s et renvoie True si elle est présente
    False dans le cas contraire
    """
    for k in range(len(ch)-len(mot)+1):
        if ch[k:k+len(mot)] == mot:
            # mot présent à la position k
            return True
    return False
```

2.4. Sur les dictionnaires

(Dictionnaires des occurences d'une liste (c'est-à-dire des effectifs de chaque élément))

```
def dico_occur(L):
    D = {}
    for x in L:
        if x not in D:
            D[x] = 1
        else :
            D[x] += 1
    return D
```

3. SCRIPTS SUR LES TRIS

3.1. Itératifs

(Tri par sélection du minimum (en place))

```
def tri_selection(L):
    """
    Trie la liste L selon le tri par sélection (du min) (en place)
    """
    n = len(L)
    for i in range(n-1):
```

```
        # Recherche du minimum de L[i:]
        mini, ind_mini = minimum_premind(L[i:])
        # On le place au début de L[i:]
        L[i], L[i + ind_mini] = L[i + ind_mini], L[i]
```

(Tri par insertion)

```
def insertion(L_tri, x):
    i = 0
    while (i < len(L_tri)) and not (L_tri[i] >= x):
        i += 1
    # insertion a la bonne place
    L_tri.insert(i, x)

def tri_insertion(L):
    """
    Trie la liste L selon le tri par insertion (Non en place)
    """
    L_tri = [L[0]]
    for x in L[1:]:
        insertion(L_tri, x)
    return L_tri
```

3.2. Récursifs

(Tri rapide (non en place))

```
def tri_rapide_rec(L):
    """
    Trie la liste L selon le tri rapide (Non en place)
    """
    if len(L) == 0:
        return L
    else:
        pivot = L[0]
        inf_pivot = []
        sup_pivot = []
        for x in L[1:]:
            if x < pivot:
                inf_pivot.append(x)
            else:
                sup_pivot.append(x)
        return tri_rapide_rec(inf_pivot) + [pivot] + \
        ↪ tri_rapide_rec(sup_pivot)
```