

Chapitre # (ANN) 1

Synthèse pour l'algorithmique

Résumé & Plan

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

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

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
<code>len(L)</code>	longueur

MANIPULATIONS DE LISTES

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

Manipulations de chaînes

Commande	Effet
<code>len(s)</code>	longueur
<code>s[0]</code>	premier élément
<code>s[-1]</code>	dernier élément
<code>s[i:j]</code>	chaîne extraite des éléments d'indices entre i (inclus) et j (exclus)
<code>s[i:]</code>	chaîne extraite à partir de l'indice i (inclus)
<code>s.join</code>	concatène à la chaîne de départ la nouvelle
<code>L = s.split()</code>	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

>_☞ (Itérateurs 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 avançant 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):
    """
    retourne la factorielle de n de manière itérative
    """
    P = 1
    for k in range(1, n+1):
        P *= k
    return P

def factorielle_rec(n):
    """
    Retourne la factorielle de n, de manière récursive
    """
    if n == 0:
        return 1
    return n*factorielle_rec(n-1)
```

2.2. Sur les listes

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

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

```
return False
```

>_☞ (Calcul d'une somme)

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

>_☞ (Calcul du produit)

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

>_☞ (Calcul du maximum)

```
def maximum(L):
    """
    Retourne 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_occure(L):
    """
    Retourne le maximum de L, et renvoie la liste des \
    ↪ occurences où il apparaît
    """
    # Recherche du maximum
```

```

maxi = L[0]
for k in range(1, len(L)):
    if L[k] > maxi:
        maxi = L[k]
# Recherche des indices
ind_maxi = []
for k in range(len(L)):
    if L[k] == maxi:
        ind_maxi.append(k)
return maxi, ind_maxi

```

```
def maximum_occur_bis(L):
```

```

"""
Retourne le maximum de L, et renvoie la liste des \
↳ occurrences où il apparaît.
Un seul parcours de la liste ici.
"""

```

```

maxi = L[0]
ind_maxi = [0]
for k in range(1, len(L)):
    if L[k] == maxi:
        ind_maxi.append(k)
    if L[k] > maxi:
        maxi = L[k]
        # découverte d'un nouveau potentiel max, on vide \
↳ la liste
        ind_maxi = [k]
return maxi, ind_maxi

```

>_🔗 (Calcul du maximum + position du premier indice)

```

def maximum_preind(L):
"""
Retourne 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 occurrences 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 & non en place))

```

from minimum import *
def tri_selection(L):
    """
    Trie la liste L selon le tri par sélection (du min) (en \
    ↪ place)
    """
    for i in range(len(L)):
        # Recherche du minimum de L[i:]
        mini, ind_mini = minimum_ind(L[i:])
        # On le place au début de L[i:]
        L[i], L[i + ind_mini] = L[i + ind_mini], L[i]

def tri_selection_nn_en_place(L):
    """
    Trie la liste L selon le tri par sélection (du min) (Non \
    ↪ en place)
    """
    L_trie = []
    for _ in range(len(L)):
        mini, ind_mini = minimum_ind(L)
        L_trie.append(mini)
        del L[ind_mini]
    return L_trie

```

>_🔗 (Tri par insertion)

```

def tri_insertion(L):
    """
    Trie la liste L selon le tri par insertion (Non en place)
    """
    L_tri = [L[0]]
    for k in range(1, len(L)):
        i = 0
        # recherche de l'indice d'insertion de L[k] dans L_tri
        while (i < len(L_tri)) and (L_tri[i] < L[k]):

```

```

        i += 1
        L_tri.insert(i, L[k])
    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 []
    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)

```