ALGORITHMIQUE

NSI | 01

RAPPELS UTILES SUR PYTHON

[Marion SZPIEG – Frédéric PEURIERE]

Création de listes

Boucles imbriquées, mesure d'un temps d'exécution

Etude d'un algorithme écrit en pseudo code

1. CREATION ET MANIPULATION DE LISTES:

Commençons avec quelques ligne de codes dans la console de THONNY:

☞ Créer de deux manières différentes une liste (`ma_liste`) contenant les nombres pairs compris entre 0 et 20 (inclus).

☞ Afficher dans la console tous les éléments de la liste.

☞ Afficher dans la console tous les éléments de la liste avec leurs indices.

☞ Afficher le premier et le denier élément de la liste.

☞ Echanger les positions du premier et du denier élément de la liste.

☞ Créer une liste (`hasard`) de 50 entiers aléatoires compris entre 0 et 100. La fonction `randint(a,b)` du module `random` renvoie un entier pseudo aléatoire compris entre a et b (inclus).

☞ Créer une fonction **`recherche(liste,nombre)`** qui permet de savoir si l'entier **`nombre`** se trouve dans la liste.

2. BOUCLES IMBRIQUEES MESURES TEMPORELLES

☞ Testez ce script: Combien de ligne affiche-t-il? .....

```
n=10
for i in range (n):
    print(i)
```

☞ Testez maintenant ce script: Combien de ligne affiche-t-il? .....

```
n=10
for i in range (n):
    for j in range(n):
        print(i,j)
```

☞ Et celui-ci? .....

```
n=10
for i in range (n):
    for j in range(n):
        for k in range (n):
            print(i,j,k)
```

On remarque que le nombre de boucles imbriquées augmente considérablement le nombre d´**opération élémentaires** (afficher la valeur des variables dans cet exemple) à effectuer et donc le **temps d´exécution**.

Pour mesurer le temps d´exécution d´un programme on utilise le module **`time`** de PYTHON:

```
from time import time
debut = time()
# code dont on mesure le temps d´execution
fin=time()
print('Temps mesuré:',fin-debut)
```

☞ Mesurer le temps d´exécution du deuxième exemple: .....

```
n=10
for i in range (n):
    for j in range(n):
        print(i,j)
```

☞ Recommençons en doublant la taille des données (n=20): .....

Le temps d´exécution dépend du **nombre de données** à traiter. On remarque que dans ce cas le temps n´est pas toujours proportionnel à ce nombre.

On parle de **`complexité temporelle`** d´un algorithme dont on essayera de donner une estimation dans les prochains chapitres.

3. ECRITURE D´UN ALGORITHME:

Quand on écrit un algorithme, on utilise un langage "naturel" ("tant que", "si"...), ce langage appelé aussi **pseudo code** permet de passer facilement à un langage de programmation on dit alors que l'on implémente l'algorithme.

Voici un premier algorithme écrit en pseudo code:

```
1 DEBUT
2    FONCTION RechercheMin(T: tableau de n entiers)
3       n ← taille du tableau T
4       min ← T[0]
5       POUR i ALLANT_DE 1 A n-1
6          SI min> T[i]
7             min← T[i]
8          FIN_SI
9       FIN_POUR
10      RETOURNE min
11 FIN
```

Regardons la séquence d´instructions pour un tableau contenant les valeurs 8, 9 , 4 et 6:

⇨ Tableau d´entier: T

indice	0	1	2	3
valeur	8	9	4	6

⇨ Initialisation (lignes 3 et 4): `n ← 4 et min ← 8`

⇨ Entrée dans la boucle (i=1): `min>T[1]` est **FAUX**

indice	~~0~~	1	2	3
valeur	8	9	4	6

⇨ i=2: `min>T[2]` est **VRAI**: `min ← 4`

indice	~~0~~	~~1~~	2	3
valeur	~~8~~	~~9~~	4	6

⇨ i=3: `min>T[3]` est **FAUX**

indice	~~0~~	~~1~~	~~2~~	3
valeur	~~8~~	~~9~~	4	6

⇨ On sort de la boucle, la valeur retournée par la fonction est **4**.

☞ Implémenter cet algorithme en langage PYTHON.