1 TD 2 : Variables, boucles, tests

(correction page ??)

Abordé lors de cette séance	
programmation	variables, boucles, tests
algorithme	recherche dichotomique

En cas de problème avec les accents, il faut ajouter au début du programme #`coding : latin − 1`.

Première demi-heure : Variables .

Un algorithme manipule des données. Ces données ne sont pas connues au moment où on écrit l'algorithme. Les variables servent à nommer ces données afin de pouvoir écrire cet algorithme. On procède la plupart du temps dans l'ordre suivant :

1. On écrit l'algorithme.
2. On affecte des valeurs aux variables.
3. On exécute l'algorithme.

Quelques exemples à essayer autour des variables :

– Les types très simples :

```
i = 3                    # entier = type numérique (type int)
r = 3.3                  # réel   = type numérique (type float)
s = "exemple"            # chaîne de caractères = type str (exemple n'est pas une variable)
n = None                 # None signifie que la variable existe mais qu'elle ne contient rien
                         # elle est souvent utilisée pour signifier qu'il n'y a pas de résultat
                         # car... une erreur s'est produite, il n'y a pas de résultat
                         # (racine carrée de -1 par exemple)
```

– Affectation et `print` :

```
v = anything         # affectation
print ( v )          # affichage
v1, v2 = 5, 6        # double affectation
```

– Print, format

```
x,y = 4,5
s = "addition"
print ( "{3} de {0} et {1} donne : {0} + {1} = {2}".format (x,y,x+y,s) )
```

Qui affiche :

```
additition de 4 et 5 donne : 4 + 5 = 9
```

– Connaître le type d'une variable :

```
print ( type ( v ) )           # affiche le type d'une variable
print ( isinstance (v, str) )  # pour déterminer si v est de type str
```

– Des tableaux :

```
c  = (4,5)               # couple de valeurs (ou tuple)
l  = [ 4, 5, 6.5]        # listes de valeurs ou tableaux
x  = l [0]               # obtention du premier élément de la liste l
y  = c [1]               # obtention du second élément
le = [ ]                 # un tableau vide
```

– A propos des tableaux :

```
l  = [ 4, 5 ]
l += [ 6 ]      # ajouter un élément
l.append ( 7 )  # ajouter un élément
l.insert (1, 8) # insérer un élément en seconde position
del l [0]       # supprimer un élément
del l [0:2]     # supprimer les deux premiers éléments
```

– Longueur d'un tableau et autres :

```
l = [ 4, 5, 6 ]
print ( len(l) )    # affiche la longueur du tableau
print ( max(l) )    # affiche le plus grand élément
s = l * 3           # création de la liste [ 4, 5, 6, 4, 5, 6, 4, 5, 6 ]
t = s [ 4:7 ]       # extraction de la sous-liste de la position 4 à 7 exclu
s [4:7] = [ 4 ]     # remplacement de cette liste par [ 4 ]
```

Seconde demi-heure : Boucles et tests .

Les boucles permettent de répéter un nombre fini ou infini de fois la même séquence d'instructions. Quelques exemples à essayer autour des boucles :

– Boucle `for` :

```
for i in range (0, 10) :   # on répète 10 fois
    print (i)              # l'affichage de i
    # ici, on est dans la boucle
# ici, on n'est plus dans la boucle
```

– Boucle `while` :

```
i = 0
while i < 10 :
    print (i)
    i += 1
```

– Interrompre une boucle :

```
for i in range (0, 10) :
    if i == 2 :
        continue           # on passe directement au suivant
    print (i)
    if i > 5 :
        break              # interruption définitive
```

– Parcours d'une liste :

```
l = [ 5, 3, 5, 7 ]
for i in range (0, len(l)) :
    print ("élément ",i, "=", l [ i ] )
```

– Parcours d'une liste sans indice :

```
l = [ 5, 3, 5, 7 ]
for v in l :
    print ("élément ", v )
```

– Parcours d'une liste sans fonction `range` mais avec indice :

```
l = [ 5, 3, 5, 7 ]
for i,v in enumerate(l) :
    print ("élément ",i, "=", v )
```

– Que fait le programme suivant :

```
l = [ 4, 3, 0, 1, 2 ]
i = 0
while l[i] != i :
    i = l[i]
print (i)            # que vaut l[i] ?
```

Les tests permettent de faire un choix : selon la valeur d'une condition, on fait soit une séquence d'instructions soit une autre :

– Un test :

```
v = 2
if v == 2 :
    print ("v est égal à 2")
else :
    print ("v n'est pas égal à 2")
```

– Un test tronqué :

```
v = 2
if v == 2 :
    print ("v est égal à 2")
```

– Plusieurs tests :

```
v = 2
if v == 2 :
    print ("v est égal à 2")
elif v > 2 :
    print ("v est supérieur à 2")
else :
    print ("v est inférieur à 2")
```

Les listes compactes

Plutôt que d'écrire :

```
l = [ ]
```

```
for i in range (10) :
    l.append( i*2+1)
```

On peut écrire :

```
l = [ i*2+i for i in range(10) ]
```

Troisième demi-heure : Recherche non dichotomique

Quelques notations abrégées :

```
l = [ i*2 for i in range(0,10) ]
print (l)  # qu'affiche l ?
```

Exercice 1 :

On veut écrire quelques instructions pour trouver la position du nombre `x = 7` dans la liste `l`.

```
l = [ 3, 6, 2 , 7, 9 ]
x = 7

# ...

print ( position )
```

Quatrième demi-heure : Recherche dichotomique

Exercice 2 :

La recherche dichotomique[1] consiste à chercher un élément `e` dans un tableau trié `l`. On cherche sa position :

- On commence par comparer `e` à l'élément placé au milieu du tableau d'indice `m`, s'ils sont égaux, on a trouvé,
- s'il est inférieur, on sait qu'il se trouve entre les indices 0 et $\texttt{m} - 1$,
- s'il est supérieur, on sait qu'il se trouve entre les indices $\texttt{m} + 1$ et la fin du tableau.

Avec une comparaison, on a déjà éliminé une moitié de tableau dans laquelle on sait que `p` ne se trouve pas. On applique le même raisonnement à l'autre moitié pour réduire la partie du tableau dans laquelle on doit chercher.

Pour aller plus loin ou pour ceux qui ont fini plus tôt

Que se passe-t-il lorsqu'on cherche un élément qui ne se trouve pas dans le tableau ?

Comparer le coût d'une recherche classique et celui d'une recherche dichotomique.

1. http://fr.wikipedia.org/wiki/Dichotomie