Fonctions, Méthodes et effets de bord

1 Une fonction modifiant une variable globale

Précédemment, nous avons introduit des fonctions qui n'agissaient que sur des variables locales. En python, il est néanmoins possible qu'une fonction modifie une variable globale.

1.1 Une fonction qui modifie une variable globale

L'usage d'une variable globale dans une fonction est pleinement autorisée. Deux cas sont à distinguer :

Si la fonction ne fait qu'appeler la variable globale : la variable globale doit être définie avant l'appel de la fonction.

```python
def test(x):
    x=x+n
    return x
n=10
print(test(1),n)
```

Remarque : ordre d'évaluation

Si la fonction doit modifier une variable globale : la variable doit être

- définie avant l'appel de la fonction ;
- déclarée dans la fonction comme étant globale, en utilisant le mot clé **global**.

```python
def test(x):
    global n
    n=x+n
    return x
n=10
print(n,test(1),n)
```

1.2 Applications

Décrire l'état du programme ci-dessous :

```python
def f1 (arg1) :
    x1 = arg1*a
    return x1

def f2 (arg2) :
    x2=arg2*a
    x2=arg2*x1
    return x2

a=14.
print(f1(3))
print(f2(3))
```

Qu'affiche le programme ci-dessous :

```python
def F(x):
    print ('variable locale : ',x)
    y=x**2
    return y

x=2
print('variable globale : ',x)
print(F(3))
```

2 Paradigmes

En informatique, il existe plusieurs façons d'envisager la programmation. Ces approches de programmations sont appelées « paradigme ».

- Programmation fonctionnelle : une fonction n'agit que sur ses variables locales.
- Programmation orientée objet : un objet agit sur un autre objet (exemple : un lave-vaisselle nettoie une assiette)

La programmation orientée objet est hors programme. Mais l'usage des attributs et méthodes qu'elle propose est d'un intérêt indéniable, que nous utiliserons quand le besoin se fait sentir.

2.1 Python et la programmation orientée objet

Spécificité

En python, *tout est objet*. Et tout objet est caractérisé par des attribus et des méthodes.

- un attribut est une propriété particulière de l'objet ;
- une méthode est assimilable à une fonction spécifique de l'objet.

- l'attribut peut être récupéré avec la syntaxe : objet.attribut
- la méthode peut être appelé avec la syntaxe : objet.méthode(argument)

python

```
>>> a=1.5+3.j # a est donc un complexe
>>> a.real
1.5
>>> a.imag
3.0
>>> a.conjugate ()
(1.5-3j)
>>> a.conjugate
<built-in method conjugate of complex object at 0x02D68938>
```

3 Méthodes sur les listes et les strings

On rappelle qu'une liste est modifiable, et qu'une chaîne ne l'est pas.

3.1 Sur les listes

Les méthodes modifient la liste à laquelle elles sont appliquées. Soit `L` une liste quelconque.

- L.`append(x)` : ajoute l'élément `x` à la fin de la liste L
- L.`pop(i)` : retire le $i^{\text{ème}}$ élément et renvoie sa valeur ($i = -1$ par défaut)
- L.`extend(T)` : concatène la liste `T` à la fin de la liste L
- L.`index(x)` : renvoie le plus petit indice de l'élément `x` si `x` est présent. Sinon : erreur.
- L.`remove(x)` : retire le premier élément `x` de la liste L, si `x` est présent. Sinon : erreur.
- L.`insert(i,x)` : insère l'élément `x` à l'indice `i` dans L. Si cet indice dépasse la taille de la liste, l'élément est ajouté à la fin.
- L.`count(x)` : renvoie le nombre de fois où l'élément `x` est présent dans la liste L
- L.`reverse()` : renverse l'ordre des éléments de la liste L
- L.`sort()` : trie les éléments de la liste L par ordre croissant
- L.`clear()` : vide la liste L de tous ses éléments
- L.`copy()` : renvoie une copie de la liste L (voir ci-dessous)

3.2 Méthodes sur les chaînes de caractères

Une chaîne de caractère est non modifiable. Les méthodes renvoient donc une nouvelle chaine, qu'il faudra affecter à une variable. On ne présente ici que celles utiles dans le cadre de ce cours. Soit `S` une chaîne de caractère quelconque.

- L.`index(x)` : renvoie le plus petit indice de l'élément `x` si `x` est présent. Sinon : erreur.
- L.`remove(x)` : retire le premier élément `x` de la liste L, si `x` est présent. Sinon : erreur.
- L.`insert(i,x)` : insère l'élément `x` à l'indice `i` dans L. Si cet indice dépasse la taille de la liste, l'élément est ajouté à la fin.
- L.`count(x)` : renvoie le nombre de fois où l'élément `x` est présent dans la liste L
- S.`isalpha()` : renvoie `True` si tous les caractères de S sont des lettres, `False` sinon.
- S.`isdigit()` : renvoie `True` si tous les caractères de S sont des chiffres, `False` sinon
- S.`upper()` : renvoie une chaîne en remplaçant les minuscules de S par des majuscules
- S.`lower()` : renvoie une chaîne en remplaçant les majuscules de S par des minuscules
- S.`find(sub)` : renvoie le plus petit indice de la chaîne de caractères `sub` recherchée dans la chaîne principale S. Cette fonction renvoie -1 si la recherche n'a pas abouti.
- S.`index(sub)` : renvoie l'indice de l'élément `x` si `x` est présent. Sinon : erreur.
- S.`count(sub)` : renvoie le nombre d'occurences de la chaîne de caractères `sub` dans la chaîne principale S
- S.`lstrip(car)` : renvoie une copie de la chaîne S où les caractères `car` situé en début de liste ont été retirés

4 Modifications indirectes des objets

4.0.1 Un objet et des références partagées

L'affectation d'une liste vers une liste n'est pas une copie du contenu, mais du référencement.

```
In [1]: L1 = [1, 2, 3]
[1, 2, 3]
In [2]: L2 = L1
[1, 2, 3]
In [3]: id(L1),id(L2)
(200563912, 200563912)
In [4]: L1.append(4)
In [5]: print(L2)
[1, 2, 3, 4]
```

Pour créer une nouvelle liste identique à l'originale mais indépendante, il faut utiliser les instructions nécessaires à la copie stricte, et/ou profonde :

- grâce à un transtypage : `copie1 = list(maListe)`
- grâce à une extraction de tous les éléments de maListe : `copie2 = maListe[:]`
- grâce à la méthode copye : `copie3 = maListe.copy()`
- avec le module copy :

```
import copy
copie4 = copy.deepcopy(maListe)
```

4.1 Mise en évidence de l'effet de bord

Cet effet secondaire (indésirable ou non) est à maitriser avec subtilité. Il apparait lors de l'usage combiné de variables évoluées (collections, images, etc.) et de fonctions.

```
1   def ordre1(M):
2       N=sorted(M)
3       return(N)
4
5   def ordre2(M):
6       M.sort()
7       N=M
8       return(N)
9
10  L=[2,1,4,3]
11  print('liste originelle : ',L)
12  print(ordre1(L))
13  print('liste originelle : ',L)
14  print(ordre2(L))
15  print('liste originelle : ',L)
```

Pour se protéger d'un effet de bord lors de l'usage d'un objet dans une fonction, il est nécessaire de travailler sur une copie de l'objet.