Cours Introduction à la Programmation Python V

(IP1 Python)

Arnaud Sangnier
sangnier@irif.fr

Mercredi 8 Novembre 2017
MIASHS et MATHS

Dans les épisodes précédents

- Présentation de certains aspects de Python 3
- Ce que nous avons vu :
  - Les données et leur type : int , str et bool
  - Les variables : affectation, lecture, modification
  - Toutes les instructions
  - Définition de fonctions
  - **Les listes à une dimension et leur utilisation basique**

Retour sur le passage des arguments

- Prenons ce programme

```
def f(a) :
  a = a+1
  return a

x = 3
y = f(x)
print(x)
print(y)
```

- Quand on appelle y = f(x), la valeur de x est donnée à f, mais f ne modifie pas x
- Ce programe affiche 3 et 4
- On pale de passage par valeurs

Que se passe-t-il avec les listes ?

- Prenons ce programme

```
def f(li) :
  li[0]=2 ;
  return li

li=[0]*3
lis=f(li)
print(li[0])
print(lis[0])
```

- Qu'affiche ce programme ? 2 et 2
- Pourquoi ? Que donne-t-on exactement comme arguments à f au moment de l'appel ?

Il faut comprendre à quoi correspondent les variables de listes

- En fait quand on fait : li=[0]*5
- Ce qui est stocké dans la variable li de type list, ce n'est pas la liste [0,0,0,0] mais c'est son 'adresse' en mémoire
- Ainsi si on veut copier une liste c'est à dire la créer deux fois on ne peut pas faire :

```
li=[0]*5
li2=li
```

- Ici les deux variables li et li2 indiquent la même liste

Retour sur l'utilisation des listes

- Quand on fait li = [0]*10 → on réserve 10 cases dans la mémoire à une certaine adresse que l'on stocke dans li
- li contient donc une adresse
- Si on fait li[i] on dit au programme d'aller voir dans la i-ème case située à l'adresse stockée dans li
- Quand on fait li2=li , li2 reçoit la même adresse que li, mais il n'y a toujours eu qu'une seule liste de créer

Exemple

- Prenons ce programme

```
a = 5
b = a
b = a +1
print(a)
print(b)
```

- a reçoit la valeur 5
- b reçoit la valeur de a qui vaut 5
- b augmente de 1 sa valeur
- à la fin a vaut 5 et b vaut 6

Exemple

- Prenons ce programme

```
li=[0]*6
li2 = li
li2[1]=3
print(li[1])
```

- li reçoit l'adresse $1 de la liste [0,0,0,0,0,0] créée
- li2 reçoit la valeur $1
- li2 modifie la deuxième valeur de la liste située à $1
- le programme affiche la valeur de li[1], comme li vaut $1 alors le programme affiche 3
- Pourquoi avoir écrit $1 ? En fait les adresses sont souvent des nombres en binaires choisi par le programme au cours de l'exécution

Exemple

- Prenons ce programme

```
li=[0]*6
li2=[0]*6
li2[1]=3
print(li[1])
```

- li reçoit l'adresse $1 de la liste [0,0,0,0,0,0] créée
- li2 reçoit la valeur $2 de la deuxième liste [0,0,0,0,0,0] créée
- li2 modifie la deuxième valeur de la liste située à $2
- le programme affiche la valeur de li[1], comme li vaut $2 alors le programme affiche 0
- Ici deux listes sont créées et elles sont par conséquent à des adresses différentes

Retour sur le passage en arguments

- Prenons ce programme

```
def f(li3) :
  li3[0]=2 ;
  return li3

li=[0]*3
lis=f(li)
print(li[0])
print(lis[0])
```

- Quand on fait l'appel f(li),on met dans la variable li3 de la fonction f l'adresse stockée dans li
- Au final dans ce programme il n'y a qu'une seule liste
- Quand on modifie une liste dans une fonction, cette liste est aussi modifiée quand on sort de la fonction

Exemple

- Prenons le programme suivant :

```
def g(li) :
  c = 0
  for i in range(0,len(li),1) :
    c = c + li[i]
    li[i]=0
  return c

lis = [1,2,3,4,5,6]
a = g(lis)
print(lis)
print(a)
```

- Il affiche [0,0,0,0,0,0] et 21
- La modification de la liste dans la fonction g se voit à l'extérieur de la fonction

Bien comprendre ce que doivent faire les fonctions

- Donc les fonctions peuvent modifier les listes passées en arguments
- Il faut faire la différence :

  1) Écrire une fonction qui prend en argument une liste et décale ses éléments de 1 vers la droite

  2) Écrire une fonction qui prend en argument une liste et renvoie une nouvelle liste qui correspond à la première liste avec les éléments décalés de un vers la droite

- Dans le cas 1) la liste donnée en arguments est modifiée
- Dans le cas 2) la liste donnée ne doit pas être changée
- En fait dans le cas 1), on peut même faire une fonction qui ne renvoie rien → une procédure qui ne fait que modifier liste

Cas 2)

- Écrire une fonction qui prend en argument une liste et renvoie une nouvelle liste qui correspond à la première liste avec les éléments décalés de un vers la droite

```
def shift(li) :
  newl=[0]*len(li)
  newl[0]=li[len(li)-1]
  for i in range(1,len(newl),1) :
    newl[i]=li[i-1]
  return newl

lis = [1,2,3,4]
lis2=shift(lis)
print(lis)
print(lis2)
```

- Ce programme affichera [1,2,3,4] et [4,1,2,3]

Cas 1)

- Écrire une fonction qui prend en argument une liste et décale ses éléments de 1 vers la droite

```
def shift(li) :
  tmp = li[len(li)-1]
  for i in range(len(li)-1,0,-1) :
    li[i]=li[i-1]
  li[0] = tmp
  return li

lis = [1,2,3,4]
lis2=shift(lis)
print(lis)
print(lis2)
```

- Ce programme affichera [4,1,2,3] et [4,1,2,3]
- Ici lis et lis2 indiquent la même liste.
- En fait le return ne sert à rien dans ce cas

Cas 1)

- Écrire une fonction qui prend en argument une liste et décale ses éléments de 1 vers la droite

```
def shift(li) :
  tmp = li[len(li)-1]
  for i in range(len(li)-1,0,-1) :
    li[i]=li[i-1]
  li[0] = tmp

lis = [1,2,3,4]
shift(lis)
print(lis)
```

- Ce programme affichera [4,1,2,3]

Cas 1) - Erreur extra classique

- Écrire une fonction qui prend en argument une liste et décale ses éléments de 1 vers la droite

```
def shift(li) :
  tmp = li[len(li)-1]
  for i in range(1,len(li),1) :
    li[i]=li[i-1]
  li[0] = tmp

lis = [1,2,3,4]
shift(lis)
print(lis)
```

FAUX

- Ce programme affichera [4,1,1,1]

Un autre problème classique

- Il faut faire la différence :

  1) Écrire une fonction qui prend en argument une liste et renverse l'ordre de ses éléments

  2) Écrire une fonction qui prend en argument une liste et renvoie une nouvelle liste qui correspond à la première liste renversée

- Dans le cas 1) la liste donnée en arguments est modifiée
- Dans le cas 2) la liste donnée ne doit pas être changée
- En fait dans le cas 1), on peut même faire une fonction qui ne renvoie rien → une procédure qui ne fait que modifier liste

Cas 2)

- Écrire une fonction qui prend en argument une liste et renvoie une nouvelle liste qui correspond à la première liste renversée

```
def reverse(li) :
  newl=[0]*len(li)
  for i in range(0,len(newl),1) :
    newl[i]=li[len(li)-1-i]
  return newl

lis = [1,2,3,4]
lis2=reverse(lis)
print(lis)
print(lis2)
```

- Ce programme affichera [1,2,3,4] et [4,3,2,1]

Cas 1)

- Écrire une fonction qui prend en argument une liste et renverse l'ordre de ses éléments

```
def reverse(li) :
  for i in range(0,len(li),1) :
    li[i]=li[len(li)-1-i]

lis = [1,2,3,4]
reverse(lis)
print(lis)
```

- Ce programme affichera [4,3,2,1]

Cas 1)

- Écrire une fonction qui prend en argument une liste et renverse l'ordre de ses éléments

```
def reverse(li) :
  for i in range(0,len(li),1) :
    li[i]=li[len(li)-1-i]

lis = [1,2,3,4]
reverse(lis)
print(lis)
```

FAUX

- Ce programme affichera [4,3,2,1]

Cas 1)

- Écrire une fonction qui prend en argument une liste et renverse l'ordre de ses éléments

```
def reverse(li) :
  for i in range(0,len(li),1) :
    li[i]=li[len(li)-1-i]

lis = [1,2,3,4]
reverse(lis)
print(lis)
```

- **Ce programme affichera [4,3,3,4]**

Cas 1)

- Écrire une fonction qui prend en argument une liste et renverse l'ordre de ses éléments

```
def reverse(li) :
  for i in range(0,len(li),1) :
    tmp=li[i]
    li[i]=li[len(li)-1-i]
    li[len(li)-1-i]=tmp

lis = [1,2,3,4]
reverse(lis)
print(lis)
```

- **Ce programme affichera [1,2,3,4] → FAUX encore**

Cas 1)

- Écrire une fonction qui prend en argument une liste et renverse l'ordre de ses éléments

```
def reverse(li) :
  for i in range(0,len(li)//2,1) :
    tmp=li[i]
    li[i]=li[len(li)-1-i]
    li[len(li)-1-i]=tmp

lis = [1,2,3,4]
reverse(lis)
print(lis)
```

- **Ce programme affichera [4,3,2,1]**

Une autre vision classique sur les listes

- On a vu que les listes simplement chaînées et les boucles étaient fortement liées
- Par exemple on peut remplir une liste li de taille k en faisant for i in range(0,k,1) et à chaque tour de boucle on accède à la position li[i]
- Mais ce n'est pas toujours aussi simple.
- Par exemple, écrire une fonction multiple qui prend en arguments une liste d'entier li et un entier k et qui renvoie une liste contenant les éléments de li multiples de k
- Ici on doit parcourt li pour chercher les multiples mais à chaque tour de boucle on ne doit pas ajouter un élément à la nouvelle liste

Solution

- Il faut utiliser pour remplir la nouvelle liste une variable p indiquant la position à remplir dans la nouvelle liste
- Au début p vaut 0, et à chaque fois que l'on met un nouvel élément à la position p, on augmente p de 1

```
def multiple(li ,k) :
  c=0
  for i in range(0 len(li),1) :
    if(li(i)%k==0) :
      c = c+1
  newL=[0]*c
  p =0
  for i in range(0 len(li),1) :
    if(li(i)%k==0) :
      newL[p] = li(i)
      p=p+1
  return newL
```