Cours Introduction à la Programmation Python III
(IP1 Python)

Arnaud Sangnier
sangnier@irif.fr

Mercredi 18 Octobre 2017
MIASHS et MATHS

Dans les épisodes précédents

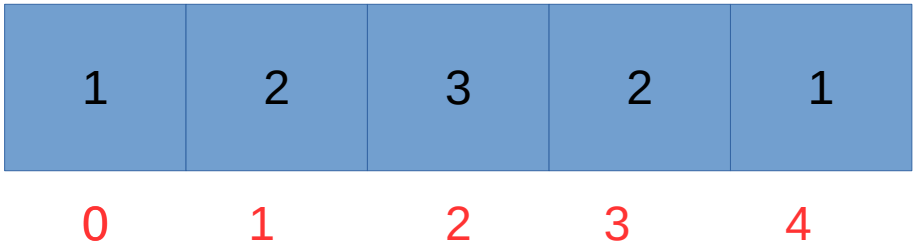
- Présentation de certains aspects de Python 3
- Ce que nous avons vu :
  - Les données et leur type : int , str et bool
  - Les variables : affectation, lecture, modification
  - Toutes les instructions :
    - manipulation de variables
    - boucles
    - tests
    - appel de fonctions
  - Définition de fonctions

Stocker des données d'une autre façon

- Imaginons qu'un programme veuille calculer la moyenne d'age d'une population de 1000 personnnes
- Pour cela il pourrait utiliser une fonction moyenne qui prendrait en paramètres 1000 entiers et renverrait la moyenne
- Deux questions :
  - Combien de variables faudrait-il ? Du coup 1000
  - Est ce que la fonction qui prendrait en paramètres 5000 entiers et renverrait la moyenne serait très différente ? Non
- On peut résoudre ces problèmes en utilisant une liste pour stocker les données

Les listes

- Une liste peut être vue comme un ensemble de cases mémoires consécutives contenant des données
- Par exemple : l = [1, 2, 3, 2, 1] est une liste que l'on pourrait représenter sous la forme



- 0, 1, 2,3 et 4 sont les indices de la liste
- C'est comme si on avait 5 variables l[0], l[1], l[2],l[3] et l[4]

Manipulation de listes

- Pour créer une liste :
  - On peut faire l = [5,6,7]
  - Mais aussi l = [0] * 1000 ← Crée une liste avec 1000 zéros
- Pour modifier la i-ème case d'une liste l, on peut faire l[i]=3
- Attention si i dépasse **la taille de la liste -1** on a un problème
- Taille de la liste : nombre cases de la liste
- Les indices vont de 0 à la taille de la liste -1
- La taille d'une liste l est donné par len(l)

Les listes → un nouveau type

- Quand on crée une liste l = [5,6,7], on obtient un nouveau type de données → list
- Qu'est ce que cela implique :
  - On a des variables indiquant des listes
  - On peut donner des listes en argument de fonctions
  - Les fonctions peuvent retourner des listes
- De plus, les éléments dans une liste peuvent être des données d'autres types
- Donc, on peut avoir:
  - des listes d'entiers
  - des listes de chaînes de caractères
  - des listes de booléens
  - mais aussi des listes de listes
  - des listes de listes de listes
  - etc

Des concepts importants sur les listes

- On va vouloir créer des méthodes qui marchent pour des listes dont on ne connaît pas la taille
- Pour cela la fonction len qui donne la taille de la liste sera nécessaire)
- Par exemple:
  - Afficher tous les éléments d'une liste
  - Chercher le plus petit élément d'une liste
  - Chercher si un élément d'une liste
  - etc
- L'utilisation de **boucles** pour parcourir la liste sera nécessaire

Parcours de listes

- L'idée d'un parcours de listes et de faire un programme qui va voir les éléments d'une liste un par un
- Par exemple,
  - si on a une liste indiquée par une variable li
  - on sait que la taille de la liste est len(li)
  - On va d'abord regarder l'élément li[0], puis li[1], puis li[2],…,jusqu'à li[len(li)-1]

```
for i in range(0,len(li),1):
  print(li[i])
```

Première fonction avec listes

- Écrire une procédure affiche qui ligne par ligne les éléments d'une liste
  - Cette fonction prend comme argument une liste, appelons la lis
  - Elle ne renvoie rien
  - Elle ne fait qu'afficher

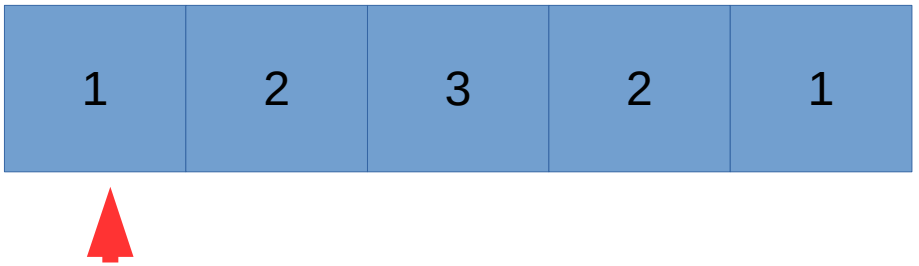
```
def affiche (lis) :
  for i in range(0,len(lis),1):
    print(lis[i])
```

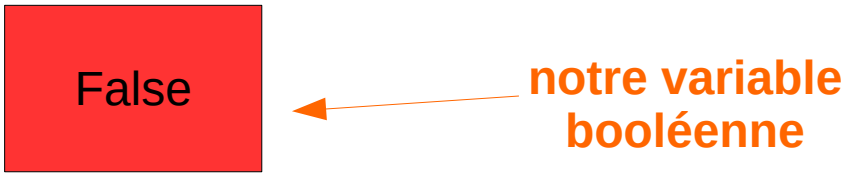
Recherche d'éléments

- On veut chercher si un élément, par exemple un entier stockée dans une variable a est dans une liste li
- Comment faire ?
  - Il faut parcourir la liste
  - Tester si un des li[i] pour i allant de 0 à len(li)-1 est égal à a
  - Mais que fait-on si c'est égal ou différent
    - Si c'est différent, il faut continuer le parcours
    - Si c'est égal, on peut continuer le parcours mais il faut se rappeler que l'on a vu l'élément, comment faire ?
    - On va utiliser une variable booléenne qui sera fausse jusqu'à ce que l'on rencontre l'élément

Intuition

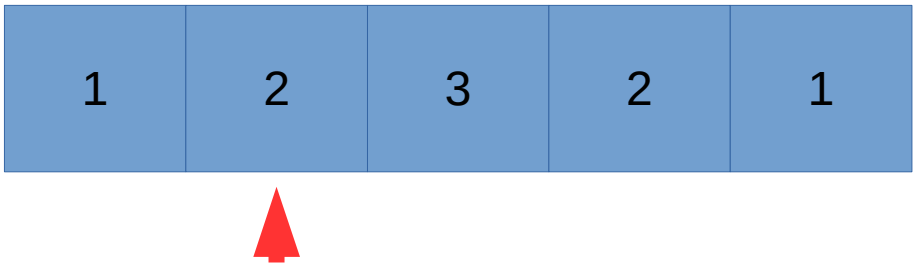
- On cherche 3 dans la liste ci-dessous

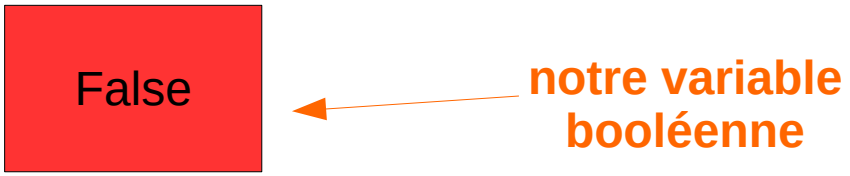




Intuition

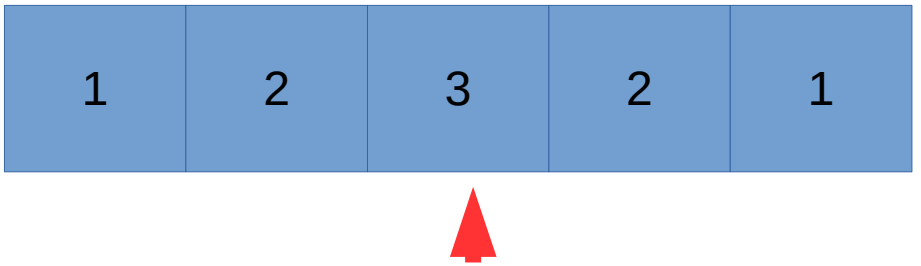
- On cherche 3 dans la liste ci-dessous

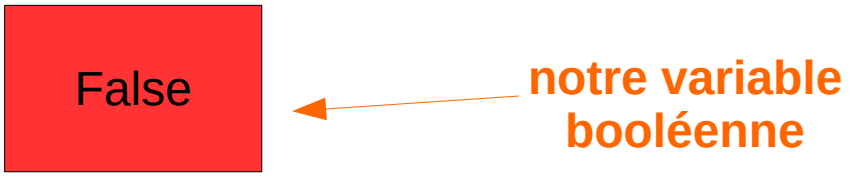




Intuition

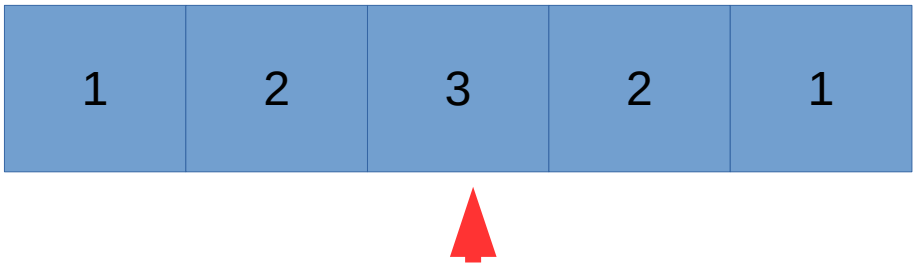
- On cherche 3 dans la liste ci-dessous
- On le trouve, la variable booléenne change

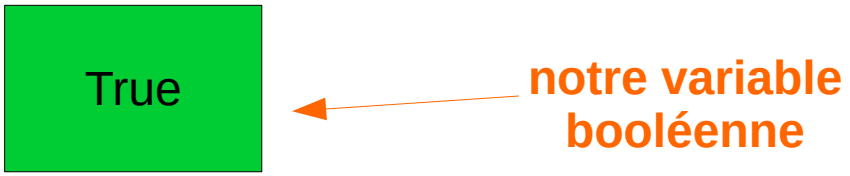




Intuition

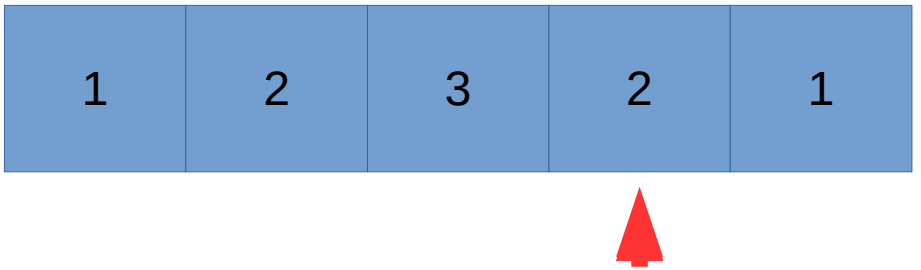
- On cherche 3 dans la liste ci-dessous
- On le trouve, la variable booléenne change

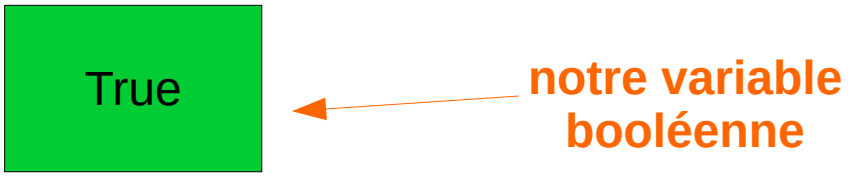




Intuition

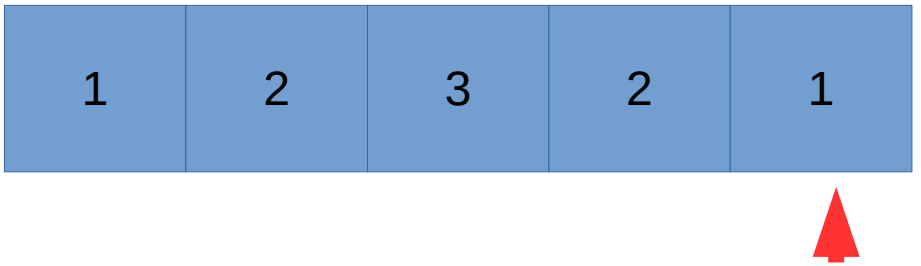
- On cherche 3 dans la liste ci-dessous

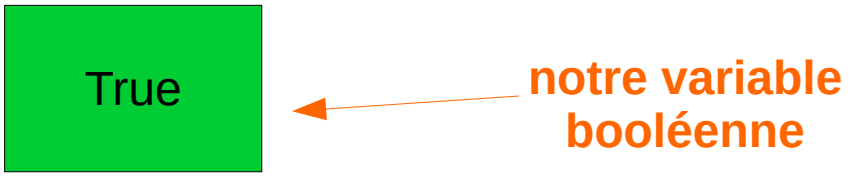




Intuition

- On cherche 3 dans la liste ci-dessous





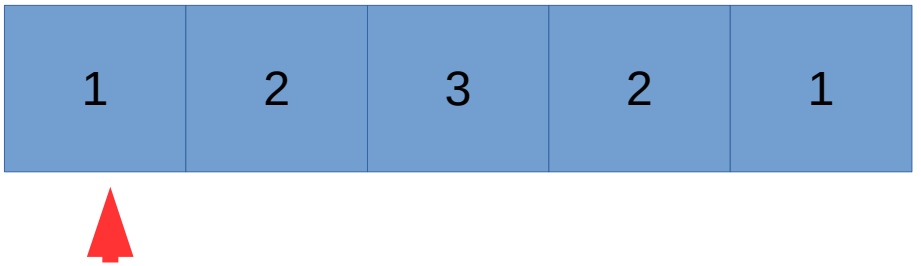
Recherche d'éléments

- Écrire une fonction qui renvoie True si un entier a est présent dans une liste d'enties li
  - Cette fonction prend deux arguments: une liste d'entiers li et un entier a
  - Elle renvoie un booléen
  - On va l'appeler cherche

```
def cherche(li, a) :
  r = False
  for i in range(0,len(li),1):
     if(li[i]==a):
        r = True
   return r
```

Erreur classique

- On cherche 3 dans la liste ci-dessous



False

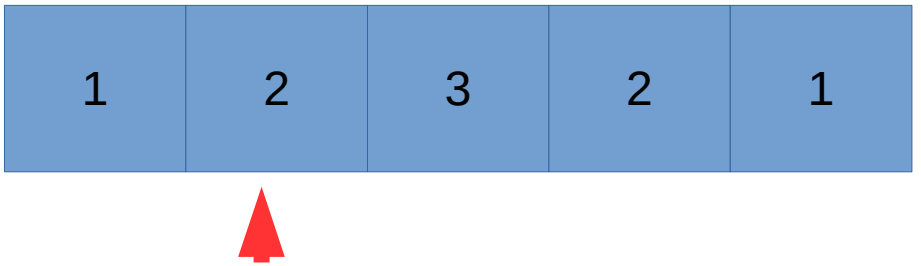
notre variable booléenne r

fausse solution !

```
def cherche(li, a) :
  r = False
  for i in range(0,len(li),1):
     if(li[i]==a):
       r = True
     else :
       r=False
   return r
```

Erreur classique

- On cherche 3 dans la liste ci-dessous



False

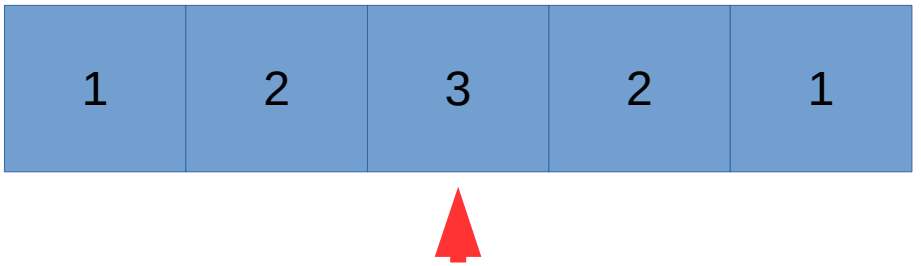
notre variable booléenne r

fausse solution !

```
def cherche(li, a) :
  r = False
  for i in range(0,len(li),1):
     if(li[i]==a):
       r = True
     else :
       r=False
   return r
```

Erreur classique

- On cherche 3 dans la liste ci-dessous



True

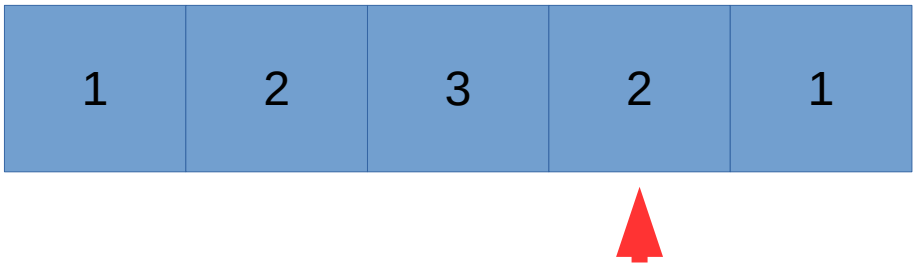
notre variable booléenne r

fausse solution !

```
def cherche(li, a) :
  r = False
  for i in range(0,len(li),1):
     if(li[i]==a):
       r = True
     else :
       r=False
   return r
```

Erreur classique

- On cherche 3 dans la liste ci-dessous



True

notre variable booléenne r

fausse solution !

```
def cherche(li, a) :
  r = False
  for i in range(0,len(li),1):
     if(li[i]==a):
       r = True
     else :
       r=False
   return r
```

Erreur classique

- On cherche 3 dans la liste ci-dessous


```
def cherche(li, a) :
  r = False
  for i in range(0,len(li),1):
     if(li[i]==a):
       r = True
     else :
       r=False
   return r
```

True

notre variable booléenne r

fausse solution !

Création de listes

- On utilise aussi des fonctions pour créer des listes
- Par exemple, une fonction seq qui crée une liste de taille n de la forme : [1,2,3,4,5,6,7,…,n]
- Cette fonction prend un argument n et renvoie une liste
- Il y a deux étapes

  1) On initialise la liste li= [0]*n

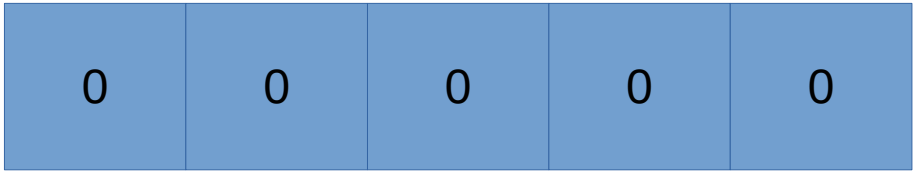
  2) On la remplit correctement avec un parcours qui change chaque élément

Intuition

- Que fait seq(5) ?

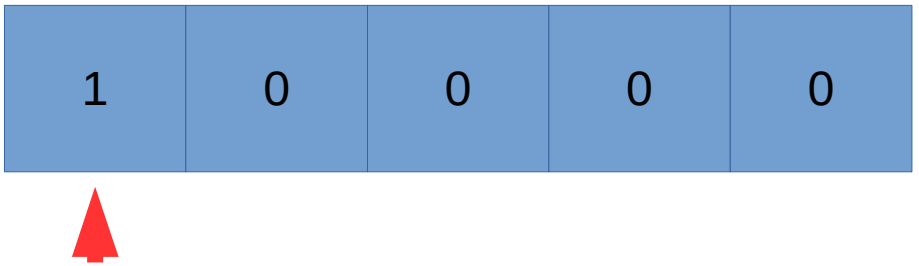
Intuition

- Que fait seq(5) ?



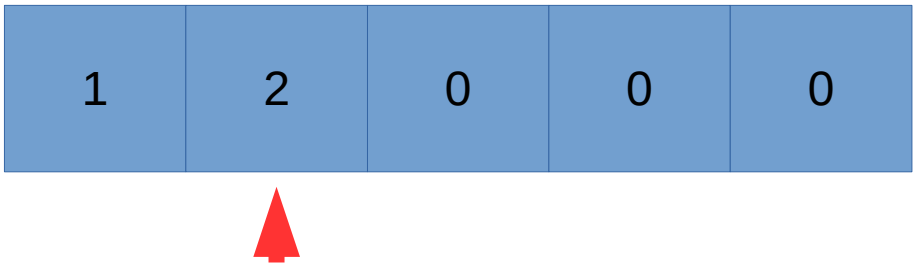
Intuition

- Que fait seq(5) ?



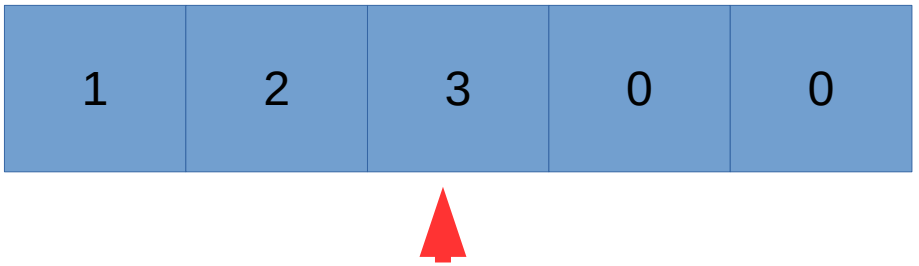
Intuition

- Que fait seq(5) ?



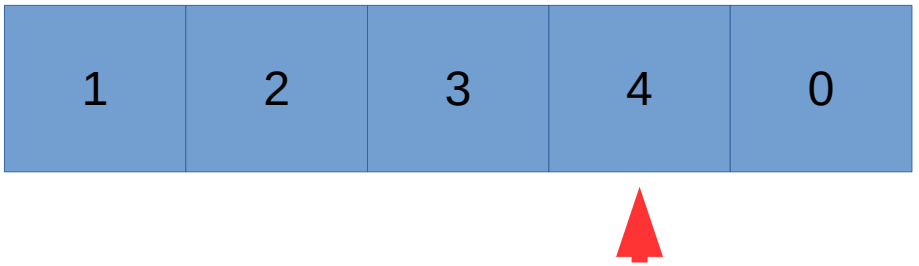
Intuition

- Que fait seq(5) ?



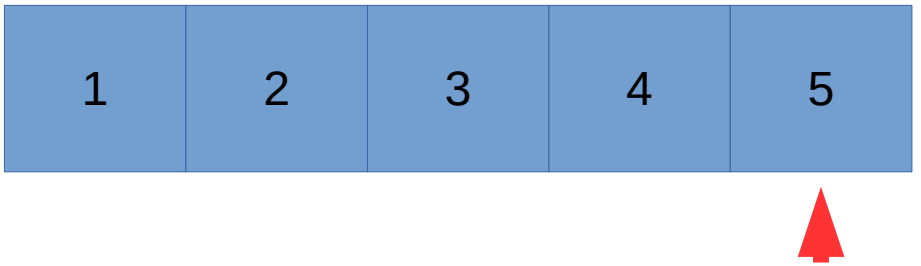
Intuition

- Que fait seq(5) ?



Intuition

- Que fait seq(5) ?



Solution

```
def seq (n) :
  li = [0]*n
  for i in range(0,len(li),1):
    li[i] = i+1
  return li
```