Les tableaux

Jeudi 3 novembre

Option Informatique

Ecole Alsacienne

Plan

1. Qu'est-ce qu'un tableau ?
2. Les tableaux en Python
3. Premières fonctions
4. Références et portée des variables
5. Diviser pour régner
6. Exercices

AVANT DE COMMENCER

- Point sur les TD
  - Merci à ceux qui ont respecté les délais
  - Pour les autres
    - Pensez à envoyer vos résultats
    - N'hésitez pas à me contacter en cas de problème

- Début de la séquence 2 : structures de données
  - Tableaux
  - Listes
  - Bac blanc

Qu'est-ce qu'un tableau ?

Plusieurs appellations

Trois façons de parler de la même chose :

- Tableau
- Vecteur
- Liste indexée

Un meuble à tiroir

QU'EST-CE QU'UN TABLEAU ?

Un tableau est un ensemble de taille n contenant des données indexées par des entiers.

- Un ensemble de taille n

- Indexée par des entiers : $[\![0 ; n-1]\!] = \{0 , 1 , \ldots , n-1\}$

Remarque : une chaîne de caractère est en quelque sorte un tableau composé de chaînes de longueur 1

B	o	n	j	o	u	r
0	1	2	3	4	5	6

Mise en situation

- Que pouvez-vous faire face à cette structure ?
- Que ne pouvez-vous pas faire face à cette structure ?

Un meuble d'une certaine taille

- La taille d'un tableau est **définie lors de la création** de ce tableau

```
t = CreerTableau(taille, valeur)
```

- Si le tableau est de taille n, il sera impossible d'accéder à son $n+1^{e}$ élément
  - Attention, les indices vont de 0 à $n-1$
  - Il n'y a donc pas d'élément d'indice n

- Cette **taille** peut être **connue** à tout moment **en temps constant** (instantanément)

```
n = Taille (t)
```

Ouvrir les tiroirs

- On peut **accéder** au contenu de n'importe quelle case en temps constant.
  - En pseudo code : `t[i]`

- On peut modifier le contenu de n'importe quelle case en temps constant
  - En pseudo code : `t[i] <- val`

- Attention, l'indexation commence à 0.

B	o	n	j	o	u	r
0	1	2	3	4	5	6

Les tableaux en Python

Définir un tableau

- Case par case :
  - Syntaxe : `nom = [ e0 , e1 , … , en ]`
  - Exemple : `t0 = [ 3 , 1 , 6 ]`

- Par sa taille et sa valeur de base :
  - Syntaxe : `nom = [valeur] * taille`
  - Exemple : `t1 = [0] * 1000`

FONCTIONS DE BASE

- Longueur du tableau :

```
longueur = len(tableau)
```

- Accéder à un élément : `t[i]`

- Modifier un élément : `t[i] = val`

- Que va afficher Python ?

```
t = [ 2 , 4 , 6 ]
print(t[3])
```

```
IndexError: list index out of range
```

Tableaux et listes en Python

- Les tableaux en Python permettent un certain nombre de choses qui ne sont pas autorisées dans d'autres langages
  - Modifier la taille d'un tableau après sa déclaration
  - Ajouter un élément au début ou à la fin
  - Supprimer un élément au milieu du tableau

- C'est en fait la même structure qui est utilisée pour représenter les tableaux et les listes

- Le type correspondant en Python est d'ailleurs `list`

Premières fonctions

Compter le nombre d'éléments pairs

- **Exercice** : Comment compter le nombre d'éléments pairs dans un tableau d'entiers ?

- **Solution** :

```
nombreElementsPairs(t) =
    n <- taille(t)
    compteur <- 0
    Pour i allant de 0 (inclus) à n (exclus)
          Faire
                Si (t[i] mod 2 = 0)
                      Alors
                            compteur <- compteur + 1
                Fin si
          Fin faire
    Renvoyer compteur
```

TROUVER LE MAXIMUM

- **Exercice** : Comment trouver le plus grand élément dans un tableau d'entiers ?

- **Solution** :

```
maxVect(t) =
    maxCourant <- t[0]
    n <- taille(t)
    Pour i allant de 1 (inclus) à n (exclus)
          Faire
                Si (t[i]>maxCourant)
                      Alors
                            maxCourant = t[i]
                Fin si
          Fin faire
    Renvoyer maxCourant
```

TROUVER L'INDICE DU MAXIMUM

- **Exercice** : Comment trouver l'indice (c'est-à-dire la position dans le tableau) correspondant à ce plus grand élément ?

- **Solution** :

```
maxVect2(t) =
    maxCourant <- t[0]
    iMaxCourant <- 0
    n <- taille(t)
    Pour i allant de 1 (inclus) à n (exclus)
          Faire
                Si (t[i]>maxCourant)
                      Alors
                            maxCourant = t[i]
                            iMaxCourant <- i
                Fin si
          Fin faire
    Renvoyer iMaxCourant
```

RECHERCHE D'UN ÉLÉMENT – VERSION BASIQUE

- **Exercice** : Comment savoir si un élément `x` apparait dans un tableau `t` ?

- **Solution** :

```
rechercheVect(t,x) =
    elementTrouve = faux
    n <- taille(t)
    Pour i allant de 0 (inclus) à n (exclus)
          Faire
                Si (t[i]=x)
                      Alors
                                elementTrouve = vrai
                Fin si
          Fin faire
    Renvoyer elementTrouve
```

Recherche d'un élément – Un peu plus malin

- **Exercice** : Comment savoir si un élément `x` apparait dans un tableau `t` ? (sans parcourir obligatoirement tout le tableau)

- **Solution** :

```
rechercheVect2(t,x) =
    elementTrouve = faux
    n <- taille(t)
    i <- 0
    Tant que ( (elementTrouve = faux) && (i<n) )
           Faire
                  Si (t[i]=x)
                         Alors
                                elementTrouve = vrai
                         Sinon
                                i = i+1
                  Fin si
           Fin faire
    Renvoyer elementTrouve
```

Références et portée des variables

COPIER UN TABLEAU

- **Question :** Que va afficher le programme suivant ?

```
t0 = [1, 2, 3]
t1 = t0
t1[0] = 4
print(t0[0])
```

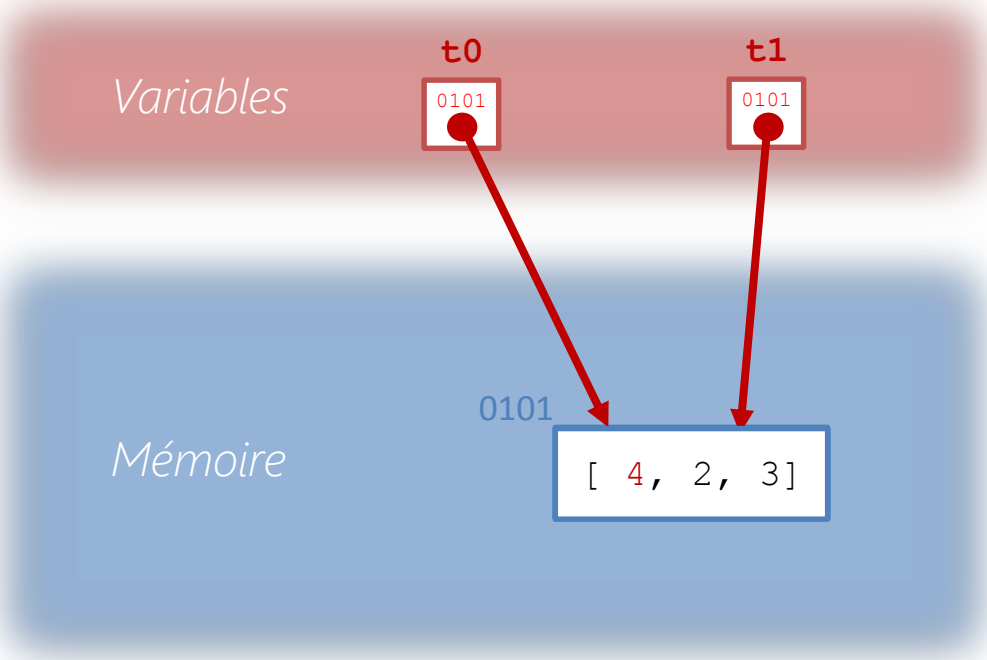
- **Réponse :** 4

- **Explication (version courte) :** Les deux variables t0 et t1 font référence au même tableau !

Copier un tableau

```
t0 = [1, 2, 3]
t1 = t0
t1[0] = 4
print(t0[0])
```

- Explication (version longue) :
  - La première instruction effectue les opérations suivantes
    - Allocation de l'espace mémoire pour stocker le tableau
    - Stockage des valeurs du tableau
    - Création d'une variable faisant référence à ce tableau
  - La deuxième instruction créé une autre variable, qui "pointe" sur le même tableau
  - La troisième instruction modifie la première case du tableau, et donc les deux variables
  - La quatrième instruction lit le contenu en mémoire



Copier un tableau – tableaux indépendants

- Pour copier un tableau en Python et obtenir deux tableaux indépendants, on utilise la fonction `list` :

```
t0 = [1, 2, 3]
t1 = list(t0)
t1[0] = 4
print(t0[0])
```

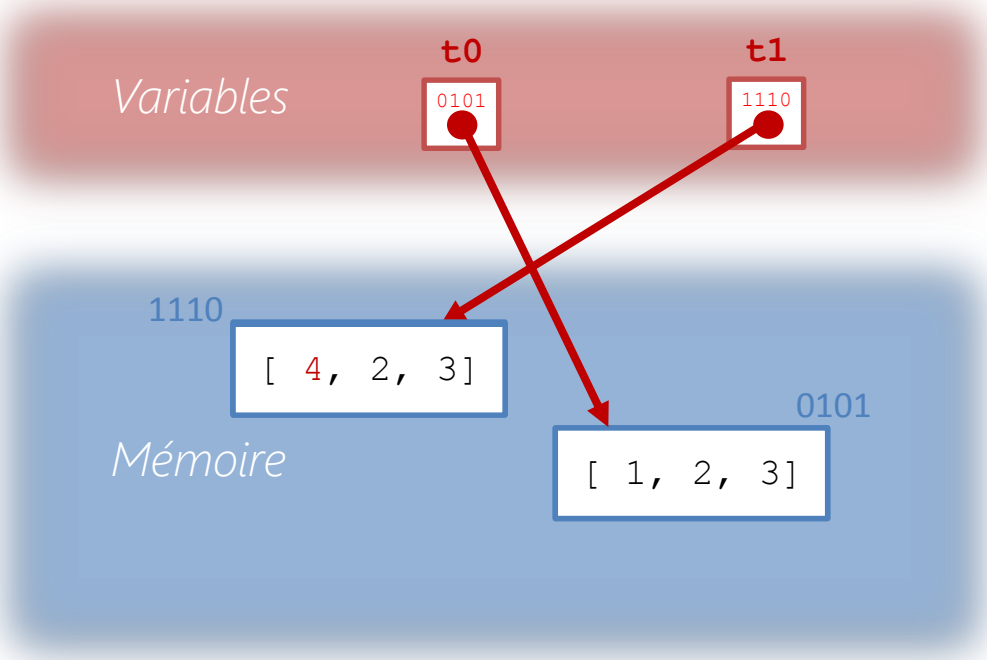
- Résultat : 1

- Remarque : On utilise ici ce qu'on appelle un "constructeur de copie"

Copier un tableau – tableaux indépendants

- Grâce à l'appel à la fonction `list`, un deuxième emplacement est alloué dans la mémoire :

```
t0 = [1, 2, 3]
t1 = list(t0)
t1[0] = 4
print(t0[0])
```



Tests d'égalité

- Python propose plusieurs opérateurs pour tester l'égalité entre deux variables
  - Le double égal permet de tester si les valeurs sont identiques
  - Le mot-clef `is` permet de tester si les références sont identiques

- Exemples

```
t0 = [1, 2, 3]
t1 = t0
print(t0 == t1)
print(t0 is t1)

True
True
```

```
t0 = [1, 2, 3]
t1 = list(t0)
print(t0 == t1)
print(t0 is t1)

True
False
```

```
t0 = [1, 2, 3]
t1 = [1, 2, 3]
print(t0 == t1)
print(t0 is t1)

True
False
```

Portée des variables

- La portée des variables est une notion importante en programmation.

- Il s'agit de répondre à la question "*Quand puis-je accéder à une variable x ?*"

- La réponse est moins triviale qu'on ne le croit

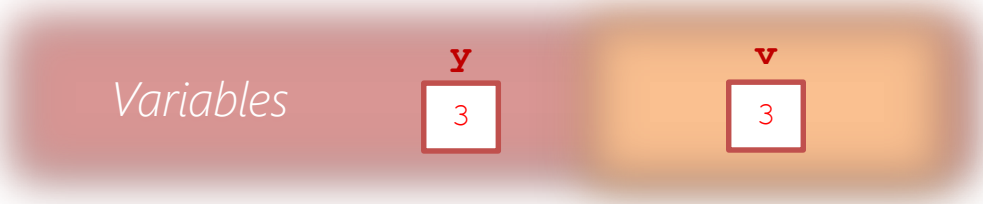
VARIABLE PASSÉE COMME ARGUMENT D'UNE FONCTION

- On a vu qu'il est possible de passer une variable comme argument d'une fonction

```
def afficherVariable(v):
    print("La variable passee en argument vaut", v)

y = 3
afficherVariable(y)
```

```
La variable passee en argument vaut 3
```

- On peut donc récupérer la valeur d'une variable définie à l'extérieur d'une fonction, à condition de la passer comme argument.



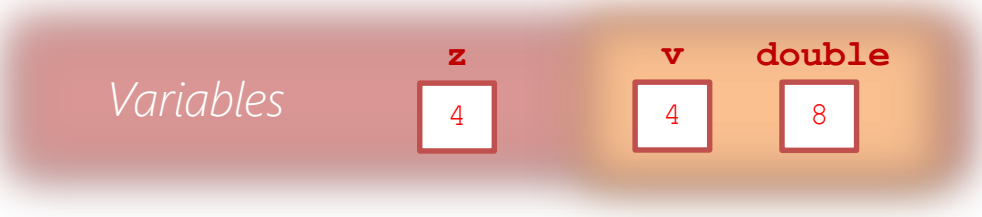
VARIABLE CRÉÉE DANS UNE FONCTION

- Il est également possible de créer des variables à l'intérieur d'une fonction :

```
def afficherVariableEtCreerDouble(v):
    print("La variable passee en argument vaut", v)
    double = v*2
    print("Le double de cette variable vaut", double)

z = 4
afficherVariableEtCreerDouble(z)
print(double)

La variable passee en argument vaut 4
Le double de cette variable vaut 8
print(double)
NameError: name 'double' is not defined
```

- Les variables définies à l'intérieur d'une fonction ne sont disponibles que dans cette fonction !

- On parle de l'espace local d'une fonction



VARIABLE DÉCLARÉE À L'EXTÉRIEUR D'UNE FONCTION

- De façon plus surprenante, il est possible d'accéder à la valeur d'une variable dans une fonction, même passer cette variable come argument

```
def afficherVariableW():
    print("La variable w vaut", w)

w = 5
afficherVariableW()

La variable w vaut 5
```

- Dans la mesure du possible, on évitera ce type de construction, car elles sont sources d'erreurs.

Modifier une variable dans une fonction

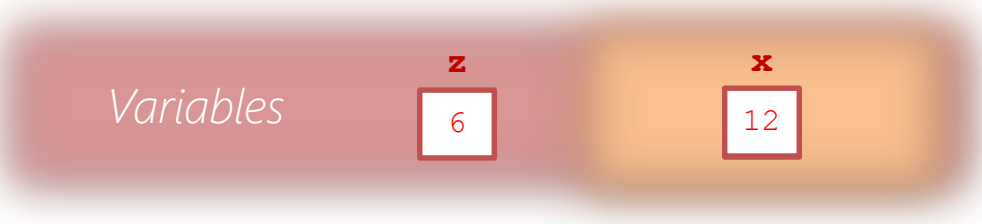
- Il est possible de modifier une variable passée en argument d'une fonction :

```
def doublerVariable(x):
    print("La variable passee en argument vaut", x)
    x = x*2
    print("A la fin de la fonction, elle vaut", x)

z = 6
doublerVariable(z)
print("Une fois la fonction terminee, la variable vaut ", z)
```

```
La variable passee en argument vaut 6
A la fin de la fonction, elle vaut 12
Une fois la fonction terminee, la variable vaut  6
```

- Cependant, ces modifications ne sont valables qu'à l'intérieur de la fonction : dès qu'on sort de la fonction, la variable retrouve son état précédent !

- Une copie de la variable passée en argument est en fait créée dans l' "espace local" de la fonction
  - C'est cette copie qu'on manipule tant qu'on est dans la fonction
  - Cette copie est détruite lorsqu'on sort de la fonction

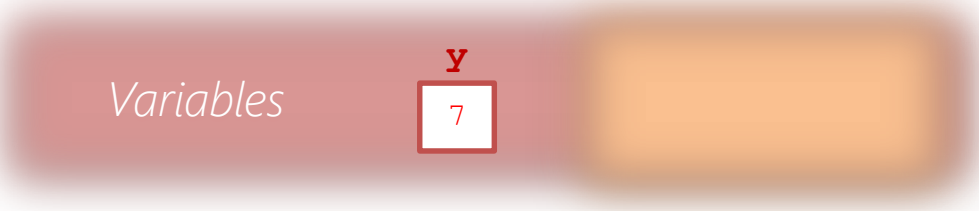


MODIFIER UNE VARIABLE DANS UNE FONCTION

- Si une variable est définie à l'extérieur d'une fonction, il n'est pas possible de la modifier dans la fonction

```
def doublerVariableY():
    print("La variable y vaut", y)
    y = y*2
    print("A la fin de la fonction, elle vaut", y)

y = 7
doublerVariableY()
print("Une fois la fonction terminee, la variable y vaut ", y)

print("La variable y vaut", y)
UnboundLocalError: local variable 'y' referenced before assignment
```

- Pour réaliser ce type d'opérations, il faudrait utiliser des variables globales (que nous essayerons d'éviter)

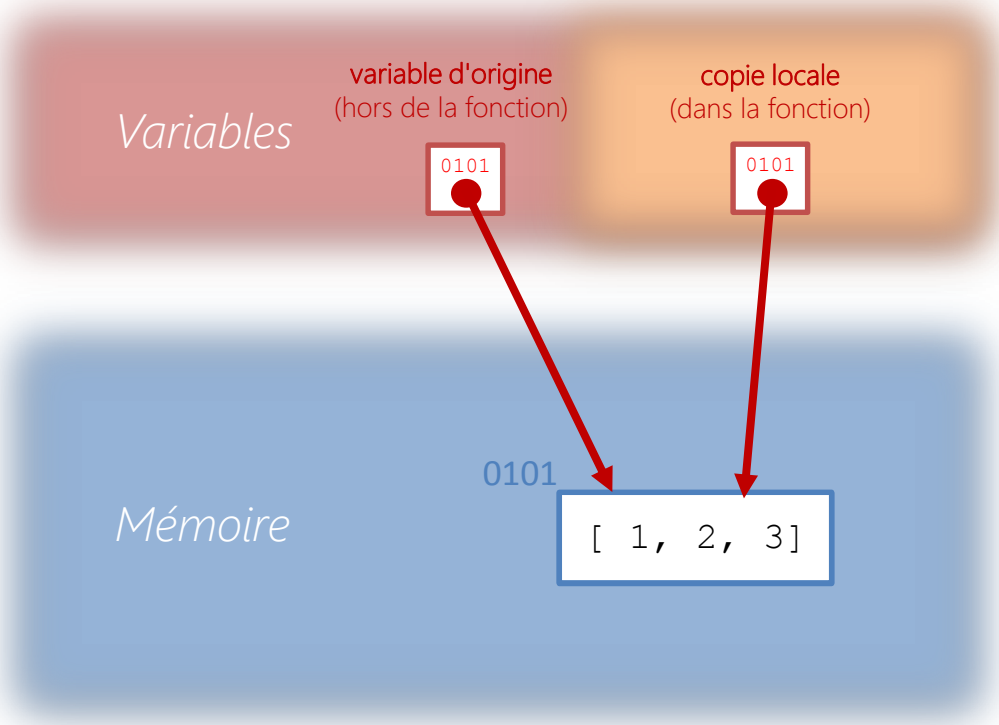


Modifier un tableau dans une fonction

- Comme on l'a vu un peu plus tôt, une variable de type `list` est en fait un pointeur sur un emplacement mémoire.

- **Conséquence 1** : La copie dans l'espace local de la fonction pointe sur le même espace mémoire que la variable d'origine

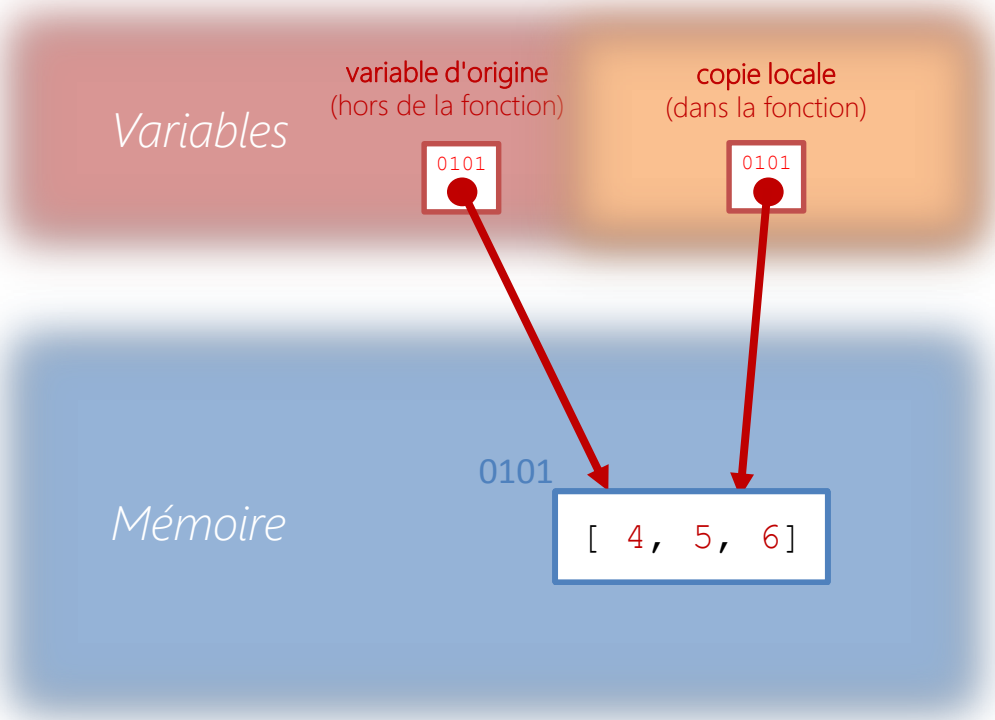
- **Conséquence 2** : Il est possible de modifier le tableau depuis l'intérieur de la fonction



Modifier un tableau dans une fonction

- Plus exactement, il est possible de modifier le tableau (case par case), mais pas de le remplacer intégralement (mise à jour de la référence)

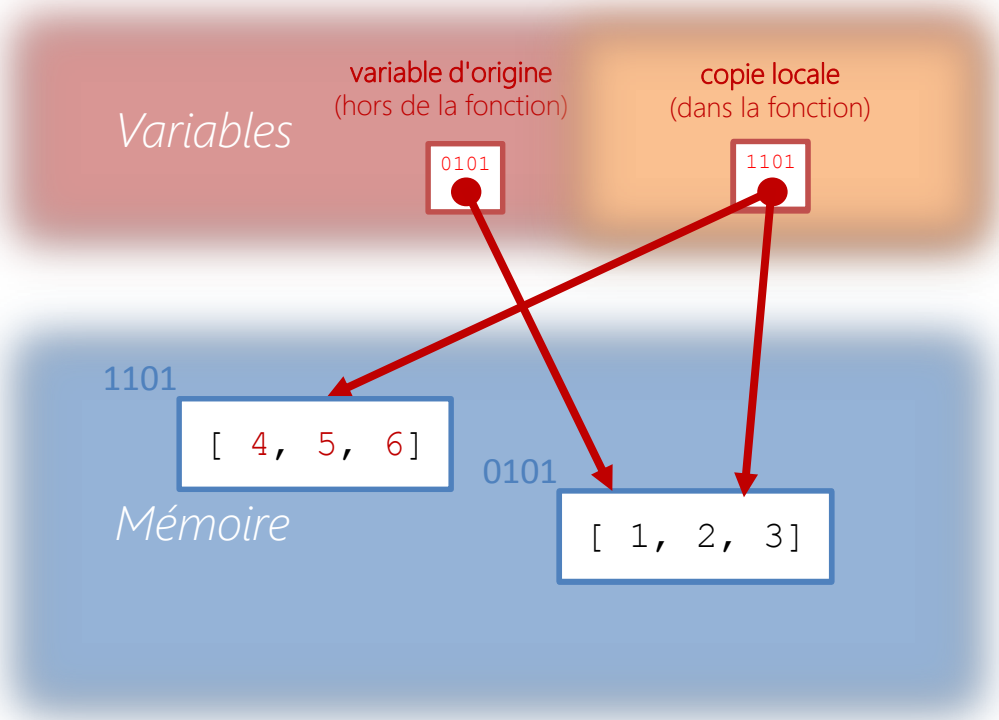
```
def modifierTab(t):
    t[0] = 4
    t[1] = 5
    t[2] = 6

t = [1,2,3]
modifierTab(t)
print(t)

[4, 5, 6]
```



Modifier un tableau dans une fonction

- Plus exactement, il est possible de modifier le tableau (case par case), mais pas de le remplacer intégralement (mise à jour de la référence)

```
def remplacerTab(t):
    t = [4, 5, 6]

t = [1,2,3]
remplacerTab(t)
print(t)

[1, 2, 3]
```



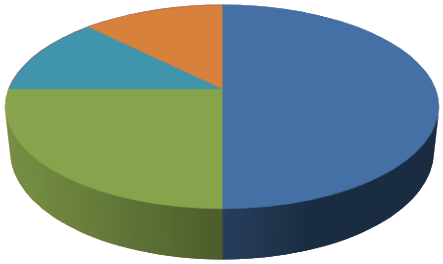
Diviser pour régner

Donnez moi un chiffre

- Donnez moi un chiffre entre 1 et 1 000

- Je vous le retrouve en 10 questions binaires (oui – non)

Comment ça marche ?



- Taille de l'intervalle de recherche :

Questions posées	0	1	2	3	4	5	6	7	8	9	10
Intervalle	1024	512	256	128	64	32	16	8	4	2	1

Diviser pour régner

- Une méthode Diviser pour régner (en anglais "divide and conquer") consiste à diviser un problème de grande taille en sous-problèmes analogues.

- Intérêt :
  - A chaque étape, on se ramène à un problème de taille plus petite
  - On finit par aboutir sur un cas de base
  - Cette approche est facile à implémenter avec les fonctions récursives

Recherche d'un élément dans un tableau trié

- **Question** : On suppose désormais que le tableau `t` est trié par ordre croissant. Comment savoir si l'élément `x` apparait dans ce tableau ?

- **Idée** : Comme **le tableau est trié**, on a pour tout $0 \leq m < n$ (n étant la taille du tableau) :
  - Soit `x=t[m]` : dans ce cas `x` appartient à `t`
  - Soit `x<t[m]` : dans ce cas `x` appartient à `t` *ssi* `x` apparait dans `t` avant l'indice `m`
  - Soit `x>t[m]` : dans ce cas `x` appartient à `t` *ssi* `x` apparait dans `t` après l'indice `m`

Recherche d'un élément dans un tableau trié

- Approche "Diviser pour régner" :
  - Soit on est dans un cas trivial : `x=t[m]`
  - Soit on se ramène à un problème analogue sur un tableau de taille plus petite :

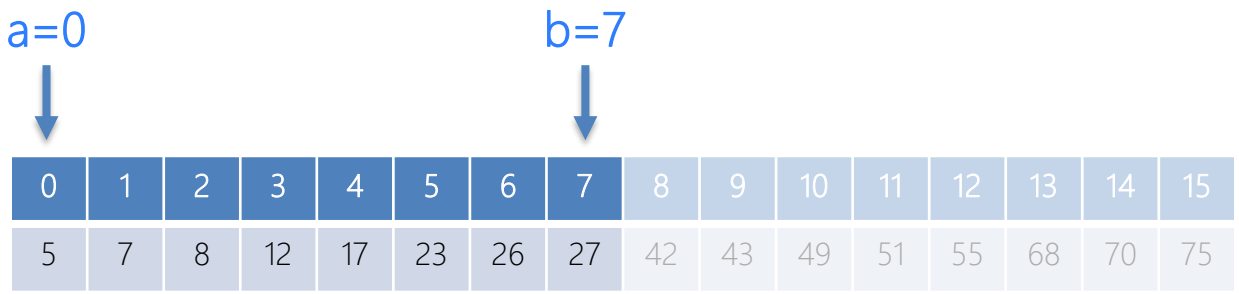
- Comment choisir `m` ?
  - On coupe l'intervalle qu'on étudie en deux
  - Plus formellement, si on travaille entre les indices `a` et `b`, on prend

```
m = (a + b) / 2
```

GÉNÉRALISER POUR MIEUX RÉSOUDRE

- Un principe assez courant en mathématiques et en informatique : on généralise le problème pour le résoudre.

- **Nouvel énoncé** : On suppose désormais que le tableau `t` est trié par ordre croissant. Comment savoir si l'élément `x` apparait dans ce tableau entre les indices a et b?



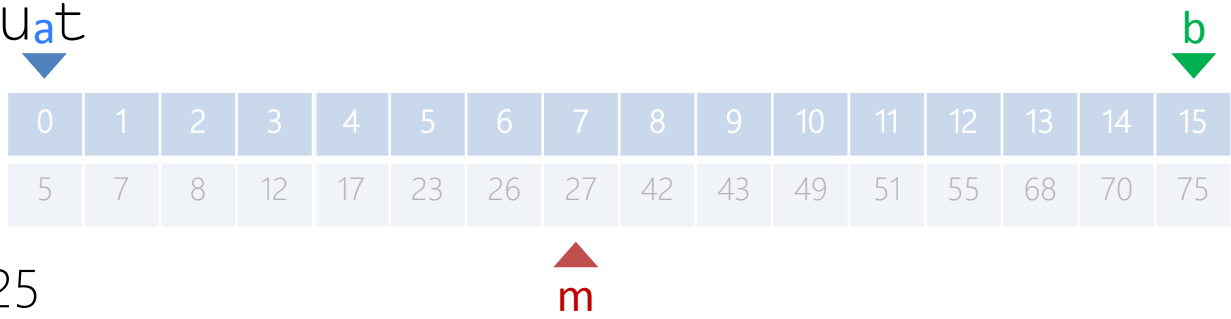
IMPLÉMENTATION

- En pseudo-code :

```
AppartientAux(x,t,a,b) =
    Si (a>b) (* Borne inférieure > Borne supérieure *)
      Alors Renvoyer faux
    Fin Si

    m <- (a+b)/2 (* milieu du sous-tableau considéré *)

    Si x=t[m]
       Alors  (* Cas de base *)
          Renvoyer vrai
       Sinon  (* Cas récursifs *)
          Si x<t[m}
             Alors Renvoyer AppartientAux(x,t,a,m-1)
             Sinon Renvoyer AppartientAux(x,t,m+1,b)
          Fin si
    Fin si
```

Quand a-t-on `a>b` ?

- Le cas `a>b` survient lorsque l'élément recherché `x` n'est pas dans le tableau `t`



- Exemple :
  - On cherche 25
  - On cherche donc :
    - Entre 0 et 15
      - `m=7`
      - `t[7] = 27`
      - recherche à gauche
    - Entre 0 et 6
      - `m=3`
      - `t[3] = 12`
      - recherche à droite
    - Entre 4 et 6
      - `m=5`
      - `t[5] = 23`
      - recherche à droite
    - Entre 6 et 6
      - `m=6`
      - `t[6] = 26`
      - recherche à gauche
    - Entre 6 et 5
      - `6>5`
      - on renvoie faux

ENCAPSULATION

- En pratique, ce qui nous intéresse, c'est de savoir si l'élément `x` est dans le tableau `t` (en entier)

- On utilise donc notre fonction `AppartientAux(x,t,a,b)` avec
  - `a = 0` (première case du tableau)
  - `b = taille(t)-1` (dernière case du tableau)

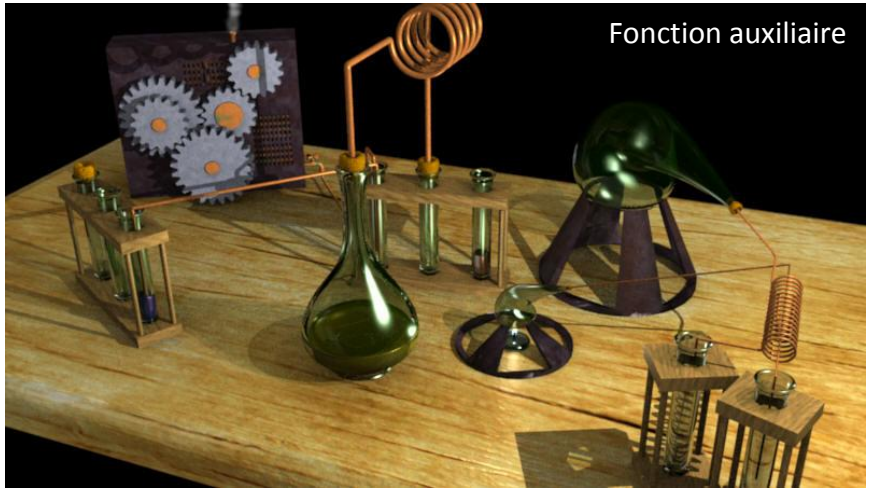
- Plutôt que de devoir taper à chaque fois ces arguments, on **encapsule** notre fonction auxiliaire dans une fonction avec moins d'arguments.

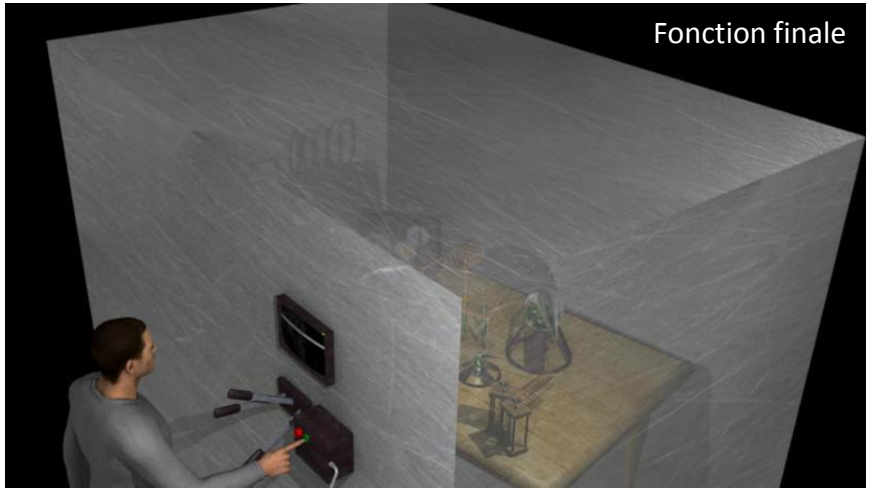
- Utilisation :

```
Appartient(t,x) =
    AppartientAux(a,b) = ...



    AppartientAux(0, taille(t)-1)
```

ENCAPSULATION

- Avantages :
  - L'utilisateur a une fonction simple à utiliser `Appartient(t,x)`
  - La fonction auxiliaire connait déjà les variables `t` et `x`, donc elle n'a que deux arguments : `AppartientAux(a,b)`
- Inconvénient :
  - La fonction auxiliaire n'est pas accessible ailleurs dans le programme





Source : Nab, Site du Zero, Tutoriel C++ sur la POO

Exercices

TRIÉ PAR ORDRE CROISSANT ?

- **Exercice** : Ecrire une fonction qui teste si un tableau est trié par ordre croissant

- **Solution** :

```
estCroissant(t) =
    valeursBienTriees <- vrai;
    n <- taille(t)
    Pour i allant de 0 à (n-2)
          Faire
                Si (t[i]>t[i+1])
                      Alors
                            valeursBienTriees = faux
                Fin si
          Fin faire
    Renvoyer valeursBienTriees
```

MOYENNE D'UN TABLEAU DE RÉELS ?

- **Exercice** : Ecrire une fonction qui calcule la moyenne d'un tableau de réels

- **Solution** :

```
moyenneVect(t) =

    somme <- 0;
    n <- taille(t)

    Pour i allant de 0 à (n-1)
          Faire
                somme <- t[i] + somme
          Fin faire

    Renvoyer (somme/n)
```

Les petits chaperons rouges

- Le petit chaperon rouge se promène dans la forêt.
- Lorsqu'elle entre sur le terrain de chasse du grand méchant loup, celui-ci la dévore.
- Heureusement, nous avons tout un stock de petits chaperons rouges en réserve
- **Question** : comment déterminer (le plus rapidement possible) où commence le terrain de chasse du Grand Méchant Loup ?

Source : Steedman, Amy. Nursery Tales. Paul Woodroffe, illustrator

Les petits chaperons rouges

- Plus formellement :
  - Forêt de taille n
  - On cherche l'indice k à partir duquel on entre sur les terres du loup

- Solution :

```
trouverLeLoup(n)=

   pas <- n/2
   endroitCalme <- 0

   Tant que (pas > 0)
   Faire
     EnvoyerChaperon(endroitCalme + pas);

     Si ChaperonEncoreEnVie()
        Alors
           endroitCalme <- endroitCalme + pas
     Fin si

     pas <- pas/2

   Fin faire

Renvoyer endroitCalme
```

- Pour approfondir :
  - Avec un seul chaperon rouge (trivial)
  - Avec uniquement deux chaperons rouges (beaucoup moins trivial)

EXPONENTIATION CLASSIQUE

- **Exponentiation** = calcul des puissances d'un nombre (en général entier)
- Notation : $a^n = a \times a \times \cdots \times a$ (n fois)
- **Exercice** : Ecrire une fonction d'exponentiation qui à partir de a et n calcule a^n
- **Solution** :

```
Exponentiation(a,n) =
Si n = 1
   Alors a
   Sinon a * Exponentiation(a, n-1)
```

PEUT-ON FAIRE PLUS RAPIDE ?

- On a suivi sans trop réfléchir la formule :

$$a^n = a \times a \times \cdots \times a \text{ (n fois)}$$

- Question : Est-ce qu'on ne peut pas faire plus rapide ?

- Idée :

$$x^n = \begin{cases} x & si\ n = 1 \\ (x^2)^{n/2} & si\ n\ est\ pair \\ x \times (x^2)^{(n-1)/2} & si\ n\ est\ impair \end{cases}$$

EXPONENTIATION RAPIDE

- **Exercice** : Ecrire une fonction d'exponentiation rapide basée sur la formule suivante :

$$x^n = \begin{cases} x & si\ n = 1 \\ (x^2)^{n/2} & si\ n\ est\ pair \\ x \times (x^2)^{(n-1)/2} & si\ n\ est\ impair \end{cases}$$

- **Solution** :

```
expRapide(x,n) =

    Si n = 1
           Alors Renvoyer x
    Fin si

    Si n est pair
           Alors Renvoyer expRapide(x*x, n/2)
           Sinon Renvoyer x * expRapide(x*x, (n-1)/2)
    Fin si
```

Exponentiations – Comparaison des résultats

- Nombre de multiplications :

N	Exponentiation classique	Exponentiation rapide
10	9	4
50	49	7
100	99	8

- Exemple :

```
er(2,10)                          appel récursif
        = er(2×2,5)               multiplication 1
        = er(4,5)                 appel récursif
        = 4×er(4×4,2)             multiplication 2
        = 4×er(16,2)              appel récursif
        = 4×er(16×16,1)           multiplication 3
        = 4×er(256,1)             cas de base
        = 4×256                   multiplication 4
        = 1024
```

Prochaine séance

Jeudi 10 novembre

[TD] Les tableaux en pratique