

LES TABLEAUX

Vendredi 16 octobre

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

PREMIERS TD À RENDRE

- Installation d'un environnement de travail chez vous
 - Merci à tous ceux qui ont joué le jeu
 - Avertissement pour les 5 élèves en retard
- Devoirs de vacances
 - TD à rendre : 1 et 2 (obligatoires tous les deux)
 - Date limite : dimanche 1^{er} novembre à 23h59
- Conseils
 - Utilisez des commentaires pour structurer votre fichier
 - Testez une dernière fois votre code en entier avant de l'envoyer
 - N'hésitez pas à envoyer un email en cas de problème

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 ?

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 : $\llbracket 0 ; n - 1 \rrbracket = \{0, 1, \dots, n - 1\}$

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

`v = CreerTableau(taille, valeur)`

- Si le tableau est de taille n , il sera impossible d'accéder à son $n + 1^{\text{e}}$ 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 (v)`

OUVRIR LES TIROIRS

- On peut **accéder** au contenu de n'importe quelle case en temps constant.
 - En pseudo code : $v[i]$
- On peut modifier le contenu de n'importe quelle case en temps constant
 - En pseudo code : $v[i] \leftarrow val$
- 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 ?

```
v = [ 2 , 4 , 6 ]
```

```
print(v[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(v) =  
  n <- taille(v)  
  compteur <- 0  
  Pour i allant de 0 à (n-1)  
    Faire  
      Si (v[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(v) =  
    max_courant <- v[0]  
    n <- taille(v)  
    Pour i allant de 1 à (n-1)  
        Faire  
            Si (v[i]>max_courant)  
                Alors  
                    max_courant = v[i]  
            Fin si  
        Fin faire  
    Renvoyer max_courant
```

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(v) =  
  max_courant <- v[0]  
  i_max_courant <- 0  
  n <- taille(v)  
  Pour i allant de 1 à (n-1)  
    Faire  
      Si (v[i]>max_courant)  
        Alors  
          max_courant = v[i]  
          i_max_courant <- i  
        Fin si  
    Fin faire  
  Renvoyer i_max_courant
```

RECHERCHE D'UN ÉLÉMENT – VERSION BASIQUE

- **Exercice** : Comment savoir si un élément x apparaît dans un tableau v ?

- **Solution** :

```
rechercheVect(v, x) =  
    il_est_la = faux  
    n <- taille(v)  
    Pour i allant de 0 à (n-1)  
        Faire  
            Si (v[i]=x)  
                Alors  
                    il_est_la = vrai  
            Fin si  
        Fin faire  
    Renvoyer il_est_la
```

RECHERCHE D'UN ÉLÉMENT – UN PEU PLUS MALIN

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

- **Solution** :

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

RÉFÉRENCES ET PORTÉE DES VARIABLES

COPIER UN TABLEAU

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

```
t0 = [1, 2, 3]
```

```
t1 = t0
```

```
t1[0] = 4
```

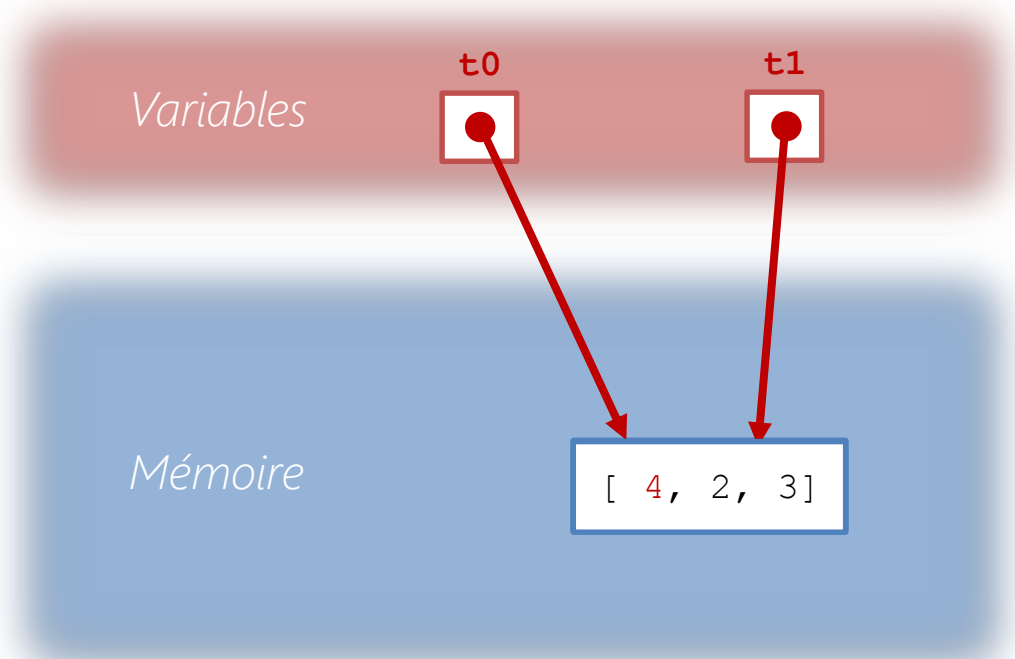
```
print(t0[0])
```

- **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 crée 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

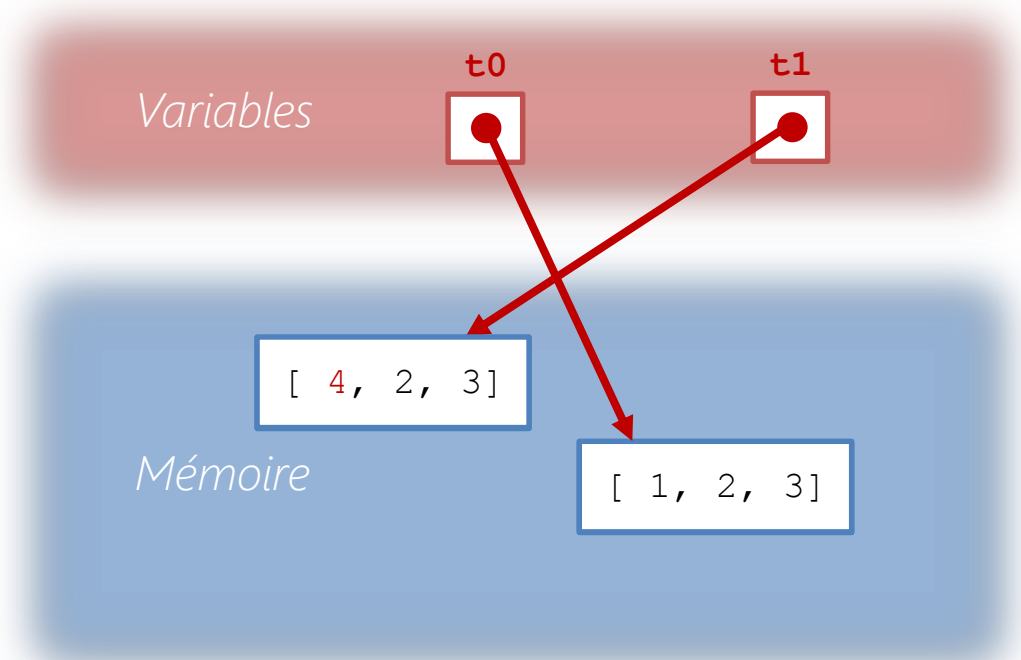
- Grâce à l'appel à la fonction `list`, un deuxième emplacement est alloué dans la mémoire :

```
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

- Exemple

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

```
y = 3  
afficher_variable(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 afficher_variable_et_creer_double(v):  
    print("La variable passee en argument vaut", v)  
    double = v*2  
    print("Le double de cette variable vaut", double)
```

```
z = 4  
afficher_variable_et_creer_double(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 afficher_variable_w():  
    print("La variable w vaut", w)
```

```
w = 5  
afficher_variable_w()
```

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 doubler_variable(x):  
    print("La variable passee en argument vaut", x)  
    x = x*2  
    print("A la fin de la fonction, elle vaut", x)  
  
x = 6  
doubler_variable(x)  
print("Une fonction la fonction terminee, la variable vaut ", x)
```

```
La variable passee en argument vaut 6  
A la fin de la fonction, elle vaut 12  
Une fonction 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 doubler_variable_y():  
    print("La variable y vaut", y)  
    y = y*2  
    print("A la fin de la fonction, elle vaut", y)
```

```
y = 7  
doubler_variable_y()  
print("Une fonction la fonction terminee, la variable y vaut ", y)
```

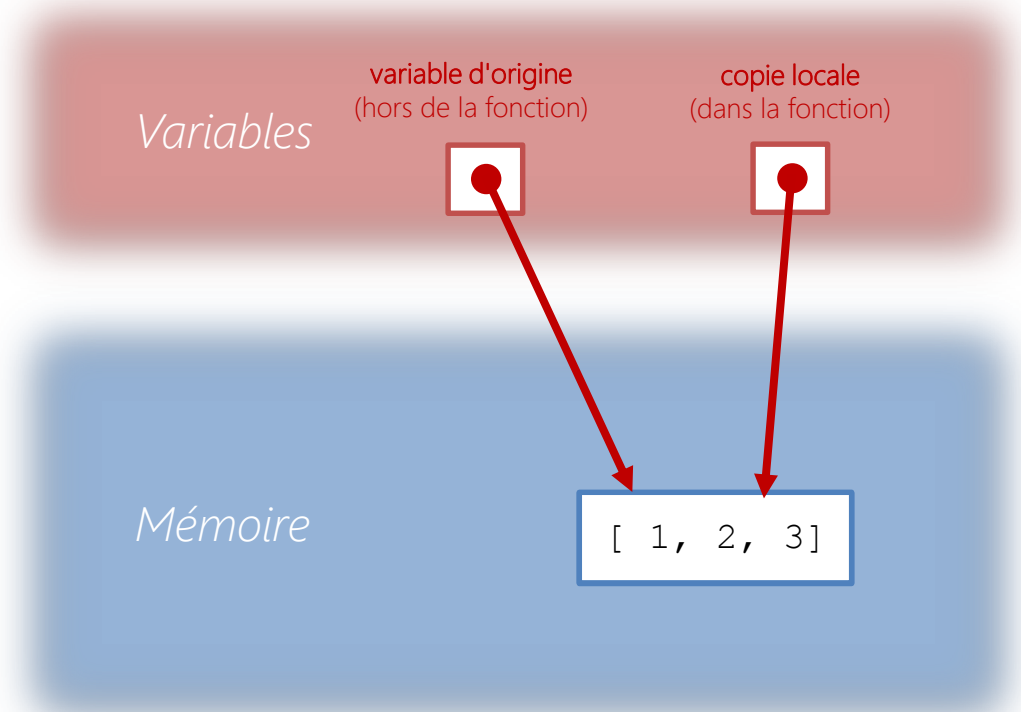
```
print("La variable y vaut", y)
```

```
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, mais pas de le remplacer intégralement

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

```
t = [1,2,3]  
modifier_tab(t)  
print(t)
```

```
[4, 5, 6]
```

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

```
t = [1,2,3]  
remplacer_tab(t)  
print(t)
```

```
[1, 2, 3]
```

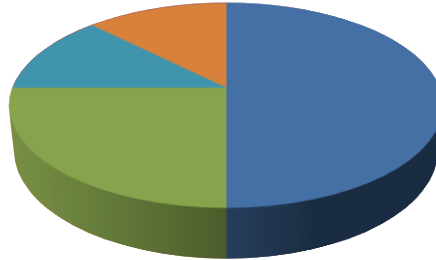
DIVISER POUR RÉGNER

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 v est trié par ordre croissant. Comment savoir si l'élément x apparaît dans ce tableau ?
- **Idée** : Comme **le tableau est trié**, on a pour tout $0 \leq m < n$ (n étant la taille du tableau) :
 - Soit $x = v[m]$: dans ce cas x appartient à v
 - Soit $x < v[m]$: dans ce cas x appartient à v ssi x apparaît dans v avant l'indice m
 - Soit $x > v[m]$: dans ce cas x appartient à v ssi x apparaît dans v après l'indice m

					m				
					v[m]				

RECHERCHE D'UN ÉLÉMENT DANS UN TABLEAU TRIÉ

- Approche "**Diviser pour régner**" :
 - Soit on est dans un cas trivial : $x = v[m]$
 - 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

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

a		m		b					

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 v est trié par ordre croissant. Comment savoir si l'élément x apparaît dans ce tableau **entre les indices a et b** ?

IMPLÉMENTATION

- En pseudo-code :

```
Appartient_Aux(x,v,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=v[m]
```

```
        Alors (* Cas de base *)
```

```
            Renvoyer vrai
```

```
        Sinon (* Cas récurifs *)
```

```
            Si x<v[m]
```

```
                Alors Appartient_Aux(x,v,a,m-1)
```

```
                Sinon Appartient_Aux(x,v,m+1,b)
```

```
            Fin si
```

```
    Fin si
```

QUAND $A > B$?

- Le cas $a > b$ survient lorsque l'élément recherché x n'est pas dans le tableau v

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	7	8	12	17	23	26	27	42	43	49	51	55	68	70	75

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

ENCAPSULATION

- En pratique, ce qui nous intéresse, c'est de savoir si l'élément x est dans le tableau v (en entier)
- On utilise donc notre fonction `Appartient_Aux(x, v, a, b)` avec
 - `a = 0` (première case du tableau)
 - `b = taille(v) - 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 :

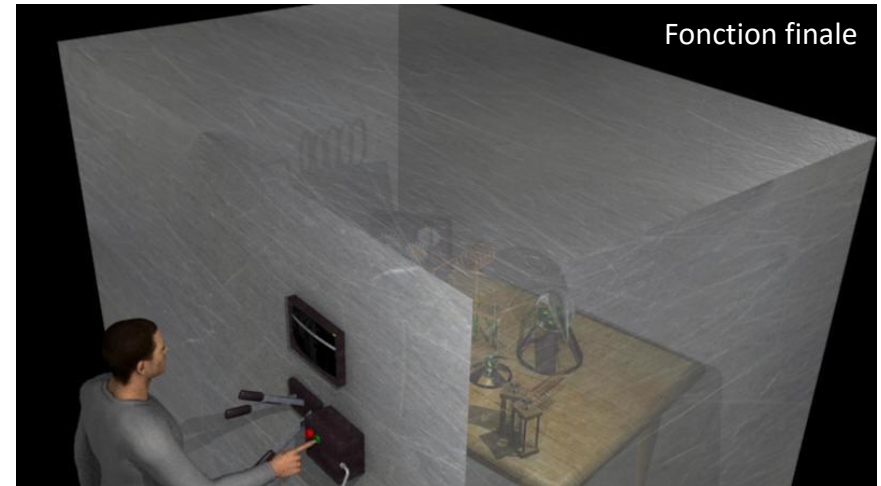
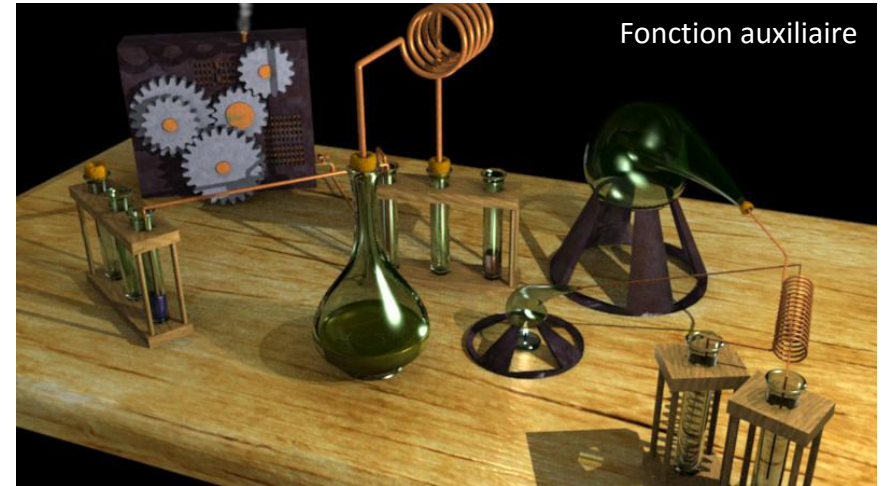
`Appartient(v, x) =`

`Appartient_Aux(a, b) = ...`

`Appartient_Aux(0, taille(v) - 1)`

ENCAPSULATION

- Avantages :
 - L'utilisateur a une fonction simple à utiliser
`Appartient(v, x)`
 - La fonction auxiliaire connaît déjà les variables `v` et `x`, donc elle n'a que deux arguments :
`Appartient_Aux(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(v) =  
  pour_l_instant <- vrai;  
  n <- taille(v)  
  Pour i allant de 0 à (n-2)  
    Faire  
      Si (v[i]>v[i+1])  
        Alors  
          pour_l_instant = faux  
        Fin si  
    Fin faire  
  Renvoyer pour_l_instant
```

MOYENNE D'UN TABLEAU DE RÉELS ?

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

- **Solution** :

```
moyenneVect(v) =
```

```
    somme <- 0;
```

```
    n <- taille(v)
```

```
    Pour i allant de 0 à (n-1)
```

```
        Faire
```

```
            somme <- v[i] + somme
```

```
        Fin faire
```

```
    Renvoyer (somme/n) (* Attention aux types *)
```

LES PETITS CHAPERONS ROUGES

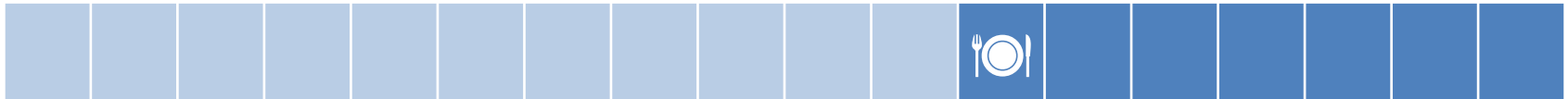
- Le petit chaperon rouge se promène dans la forêt.
- Lorsqu'elle entre sur le terrain de chasse du grand méchant loup, celui-ci la dévore.
- Heureusement, nous avons tout un stock de petits chaperons rouges en réserve
- **Question** : comment déterminer (le plus rapidement possible) où commence le terrain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

```
    endroit_calme <- 0
```

```
    Tant que (pas > 0)
```

```
    Faire
```

```
        EnvoyerChaperon(endroit_calme + pas);
```

```
        Si ChaperonEncoreEnVie
```

```
            Alors
```

```
                endroit_calme <- endroit_calme + pas
```

```
        Fin si
```

```
        pas <- pas/2
```

```
    Fin faire
```

```
    Renvoyer endroit_calme
```

- 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 : $a^n = a \times a \times \dots \times a$ (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 :

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

EXPONENTIATION RAPIDE

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

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

- **Solution** :

`expRapide(x, n) =`

`Si n = 1`

`Alors Renvoyer x`

`Fin si`

`Si n est pair`

`Alors Renvoyer expRapide(x*x, n/2)`

`Sinon Renvoyez 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 :

$\text{er}(2, 10)$

$$\begin{aligned} &= \text{er}(2 \times 2, 5) \\ &= \text{er}(4, 5) \\ &= 4 \times \text{er}(4 \times 4, 2) \\ &= 4 \times \text{er}(16, 2) \\ &= 4 \times \text{er}(16 \times 16, 1) \\ &= 4 \times \text{er}(256, 1) \\ &= 4 \times 256 \\ &= 1024 \end{aligned}$$

appel récursif
multiplication 1
appel récursif
multiplication 2
appel récursif
multiplication 3
cas de base
multiplication 4

PROCHAINE SÉANCE

Vendredi 6 novembre

[TD] LES TABLEAUX EN PRATIQUE



Et d'ici là, bonnes vacances !