

# IFT1015 Programmation 1

## Données structurées

### (structures et tableaux)

Marc Feeley

(avec ajouts de Aaron Courville et Pascal Vincent)



# Données structurées

- On a souvent besoin de faire des traitement sur des **groupes de données** :
  - Liste des étudiants d'une classe (plusieurs étudiants, plusieurs propriétés pour chaque étudiant tel le nom, la note de chaque travail, ...)
  - Groupe d'objets dans un jeu video
  - Ensemble des fichiers sur le disque dur
  - Statistiques de température à Montréal pour chaque jour de l'année dernière
  - La séquence de caractères dans un document



# Données structurées

- Pour faciliter le traitement de ces données, il faut les **organiser suivant une certaine structure**
- De façon générale, les langages de programmation offrent 2 types de donnée prédéfinis pour les **données structurées** :
  - **Enregistrement/structure** ("*record/structure*")
  - **Tableau** ("*array*")
- En JS, on a un seul type qui couvre les deux cas; le type **objet** ("*object*")
- Un tableau est un **cas spécial d'objet**



# Données structurées

- **Enregistrement/structure** : groupe de données qui sont accessibles par un **nom** indiquant la donnée voulue (on parle de **champ** ou de **propriété**)
- Les champs peuvent être de **types différents**
- Exemple : un dossier médical contient des champs :
  - **Nom du patient**, un texte
  - **Date de naissance**, un texte ou 3 entiers
  - **Genre**, un booléen ou un texte
  - **Nom du médecin de famille**, un texte
  - ...



# Données structurées

- **Tableau** : groupe de données qui sont accessibles par un **index** numérique indiquant la position dans le tableau (on parle des **éléments** du tableau)
- Exemple : température de chaque jour de 2014 (sur les 365 éléments, on pourrait être intéressé au 5ieme, la température du 5 janvier 2014, ou au 365ieme, la température du 31 décembre 2014)
- Un tableau a donc une **longueur** (le nombre d'éléments)
- En JS et la majorité des langages **le premier élément est à la position 0**



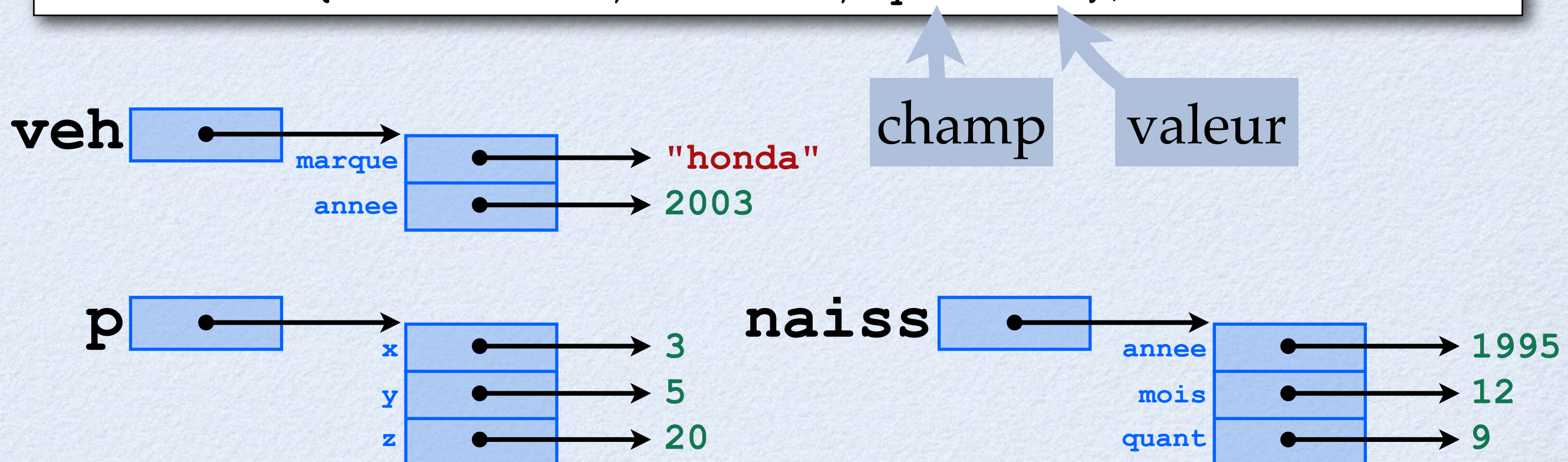
# Enregistrements

- Un enregistrement peut être construit en énumérant tous les champs entre « { » et « } », où chaque champ est une **valeur** préfixée par un **nom**

```
var veh = { marque: "honda", annee: 2003 };
```

```
var p = { x: 3, y: 5, z: 2*10 };
```

```
var naiss = { annee: 1995, mois: 12, quant: 9 };
```

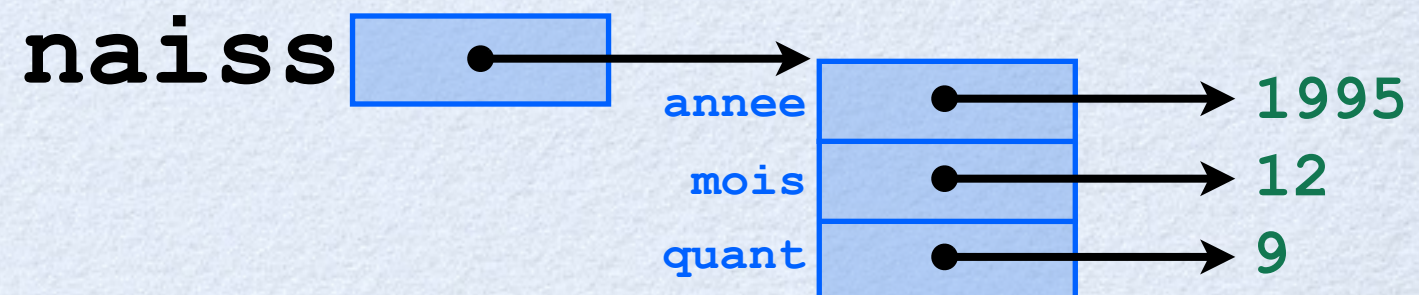
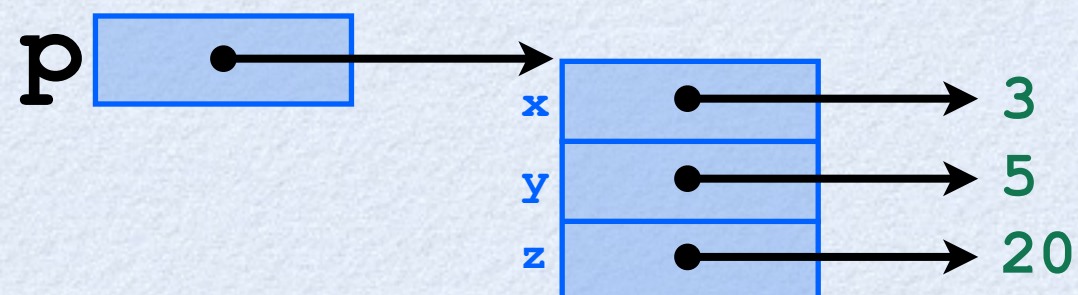
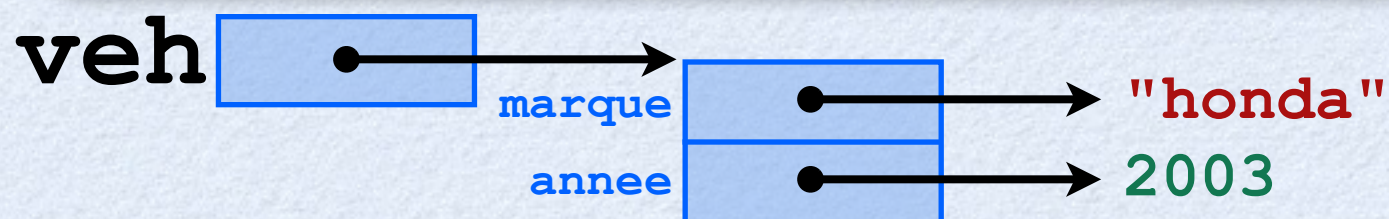




# Enregistrements

- codeBoot utilise la même syntaxe pour afficher les valeurs qui sont des enregistrements

```
> var veh = { marque: "honda", annee: 2003 };  
> var p = { x: 3, y: 5, z: 2*10 };  
> var naiss = { annee: 1995, mois: 12, quant: 9 };  
> veh  
  {marque: "honda", annee: 2003}  
> p  
  {x: 3, y: 5, z: 20}  
> naiss  
  {annee: 1995, mois: 12, quant: 9}
```





# Enregistrements

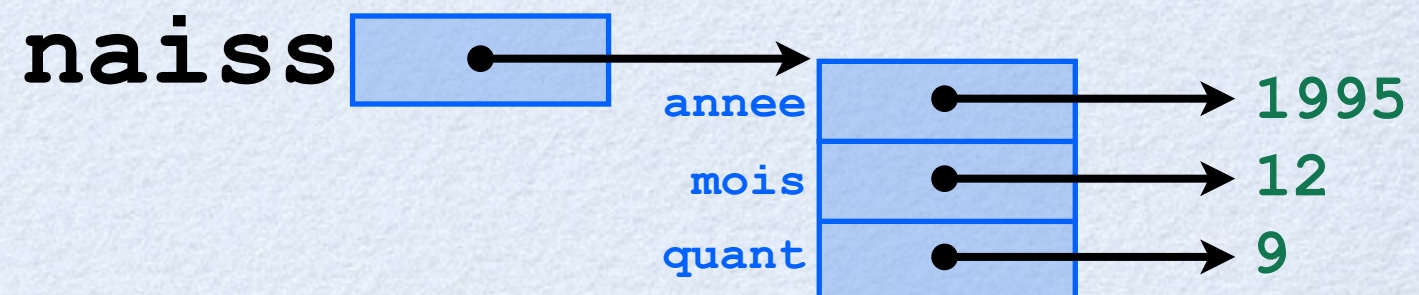
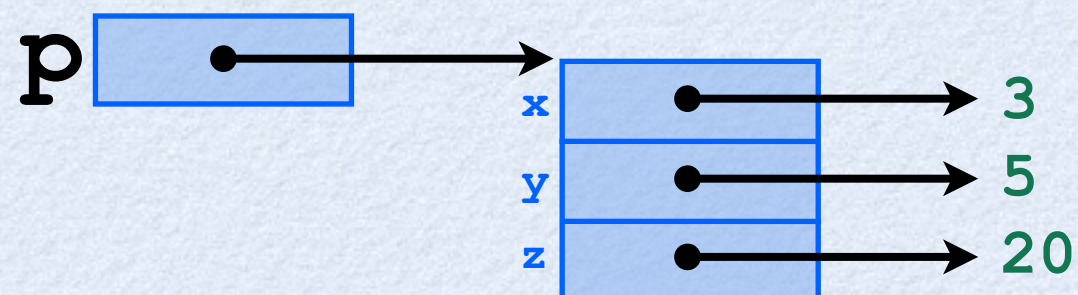
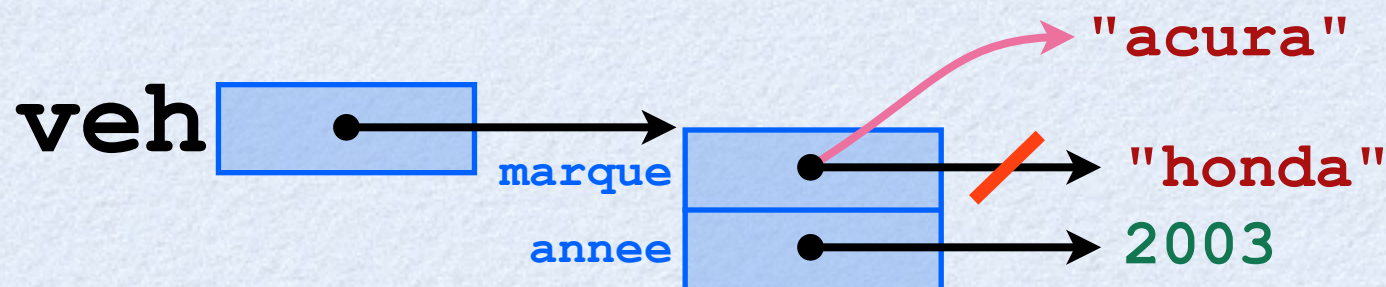
- L'accès à un champ (lecture ou écriture) se fait avec la syntaxe :  $\langle expression \rangle . \langle identificateur \rangle$

```
var veh = { marque: "honda", annee: 2003 };  
var p = { x: 3, y: 5, z: 2*10 };  
var naiss = { annee: 1995, mois: 12, quant: 9 };
```

```
print( veh.marque ); // imprime honda  
veh.marque = "acura";  
print( veh.marque ); // imprime acura
```

enregistrement

champ

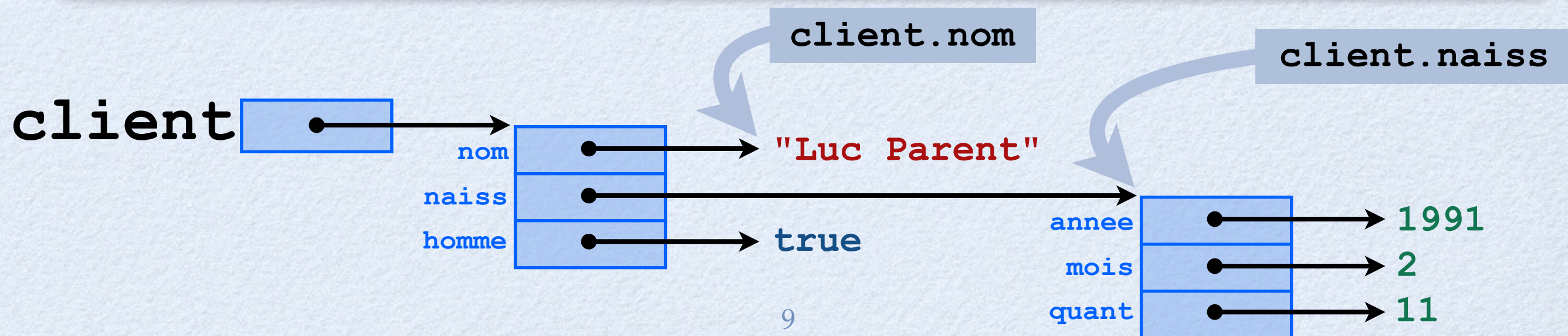




# Enregistrements

- Les champs peuvent contenir n'importe quel type incluant des **objets** (enregistrements et tableaux)

```
var client = { nom: "Luc Parent",  
               naiss: { annee: 1991,  
                       mois: 2,  
                       quant: 11 },  
               homme: true };  
  
print( client.nom );           // imprime Luc Parent  
print( client.naiss.annee );  // imprime 1991
```

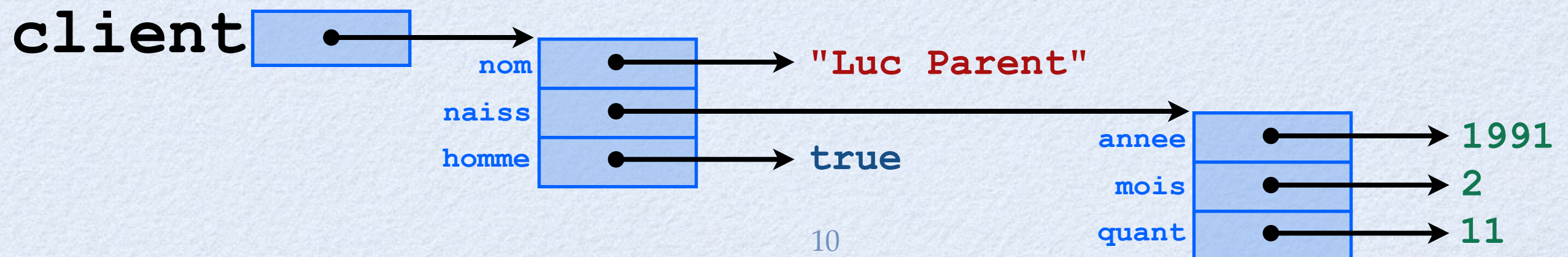




# Enregistrements

- L'affichage de structures imbriquées (structure dans une structure) se fait avec la même syntaxe

```
> var client = {...};  
> client  
{nom: "Luc Parent", naiss: {annee: 1991, mois: 2, quant: 11},  
homme: true}
```





# Exemple : points 2D

```
// Calcul du périmètre d'un triangle avec coins aux  
// coordonnées (x1,y1), (x2,y2), (x3,y3)
```

```
var perimetre = function (x1, y1, x2, y2, x3, y3) {  
    return distance(x1, y1, x2, y2) +  
           distance(x2, y2, x3, y3) +  
           distance(x3, y3, x1, y1);  
};
```

```
var distance = function (x1, y1, x2, y2) {  
    return hypotenuse(x2-x1, y2-y1);  
};
```

```
var hypotenuse = function (base, hauteur) {  
    return Math.sqrt(carre(base) + carre(hauteur));  
};
```

```
var carre = function (x) {  
    return x*x;  
};
```

```
print(perimetre(0, 0, 4, 0, 4, 3)); // imprime 12
```

c'est lourd de  
manipuler  
séparément  
chaque  
composante  
des points



# Exemple : points 2D

```
// Calcul du périmètre d'un triangle avec coins aux  
// points p1, p2 et p3.
```

```
var perimetre = function (p1, p2, p3) {  
    return distance(p1, p2) +  
           distance(p2, p3) +  
           distance(p3, p1);  
};
```

```
var distance = function (p1, p2) {  
    return hypotenuse(p2.x-p1.x, p2.y-p1.y);  
};
```

```
var hypotenuse = function (base, hauteur) {  
    return Math.sqrt(carre(base) + carre(hauteur));  
};
```

```
var carre = function (x) {  
    return x*x;  
};
```

```
print(perimetre({x:0,y:0}, {x:4,y:0}, {x:4,y:3}));
```

on remplace  
chaque paire de  
coordonnées  
par un  
enregistrement

accès aux  
champs

création  
des points



# Enregistrements comme type

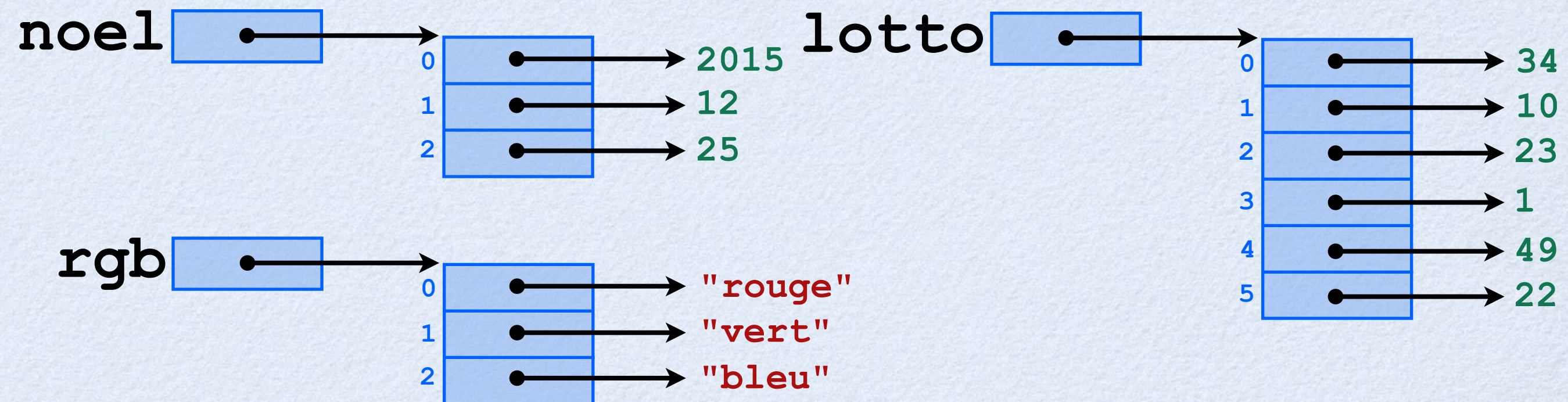
- Les enregistrements permettent de regrouper un **ensemble de données qui sont reliées** en une seule valeur
- C'est donc une façon pour le programmeur de définir un **nouveau type de donnée** :
  - Type **point 2D** (coordonnées x et y)
  - Type **personne** (nom, date de naiss., genre, ...)
  - Type **police de caractères** (nom, taille, style)
  - Type **fenêtre** (titre, taille, position)
  - Type **couleur** (composantes rouge / vert / bleu)
  - ...



# Tableaux

- Un tableau peut être construit en énumérant tous ses éléments entre « [ » et « ] »

```
var noel = [2015, 12, 25];  
  
var rgb = ["rouge", "vert", "bleu"];  
  
var lotto = [34, 10, 23, 1, 49, 22];
```



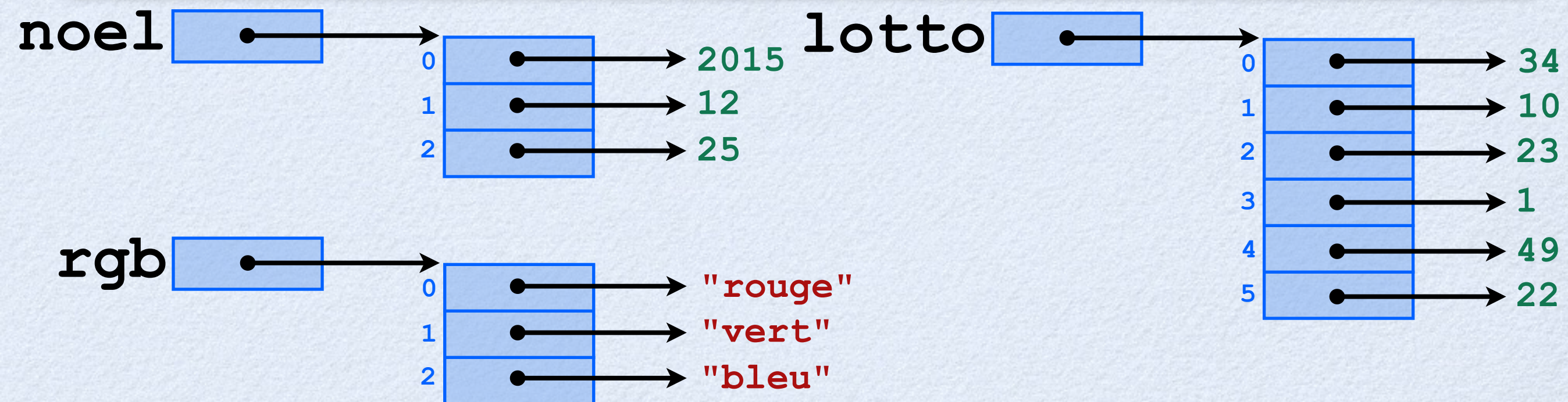


# Tableaux

- codeBoot utilise la même syntaxe pour afficher les valeurs qui sont des tableaux

```
> var noel = [2015, 12, 25];  
> var rgb = ["rouge", "vert", "bleu"];  
> var lotto = [34, 10, 23, 1, 49, 22];  
> lotto  
[34, 10, 23, 1, 49, 22]  
> lotto+""  
"34,10,23,1,49,22"
```

conversion à  
texte n'a pas de  
[ et ] et pas  
d'espaces





# Tableaux

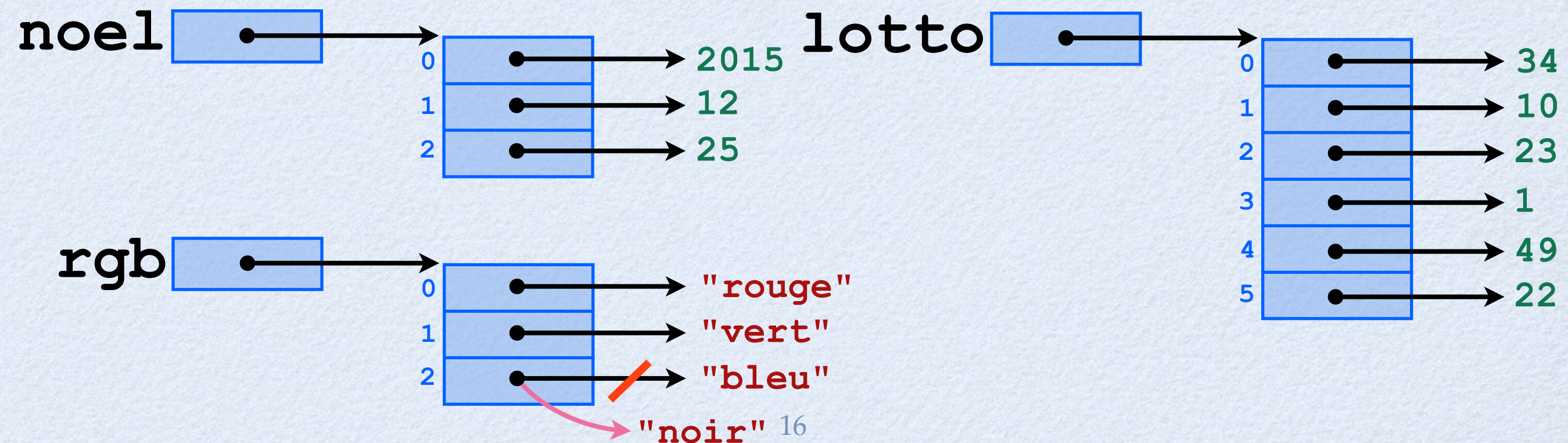
- L'accès à un élément, l'**indexation**, se fait avec la syntaxe suivante :  $\langle expression \rangle [\langle expression \rangle]$

```
var noel = [2015, 12, 25];  
var rgb = ["rouge", "vert", "bleu"];  
var lotto = [34, 10, 23, 1, 49, 22];
```

```
print( rgb[2] ); // imprime bleu  
rgb[2] = "noir";  
print( rgb[2] ); // imprime noir
```

tableau

index entier

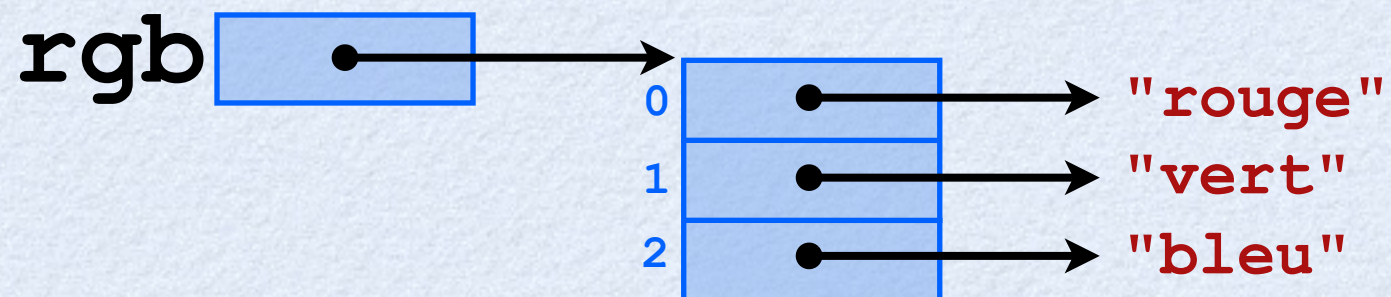




# Tableaux

- codeBoot fait des **vérifications de bornes** de l'index
- L'index doit être un entier  $\geq 0$  et  $< \text{nb. éléments}$

```
> var rgb = ["rouge", "vert", "bleu"];  
> rgb[0]  
"rouge"  
> rgb[-1]  
array index is out of bounds  
> rgb[3]  
array index is out of bounds  
> rgb[true]  
array index must be an integer
```





# Tableaux

- Un tableau peut également être construit avec son **constructeur**, la fonction **Array**, en lui passant comme unique paramètre le nombre d'éléments voulus (qui seront tous initialisés à **undefined**)
- La longueur d'un tableau s'obtient avec la syntaxe suivante : *<expression>* . **length**

tableau



```
var temp = Array(365); // températures de 2014

print( temp.length ); // imprime 365

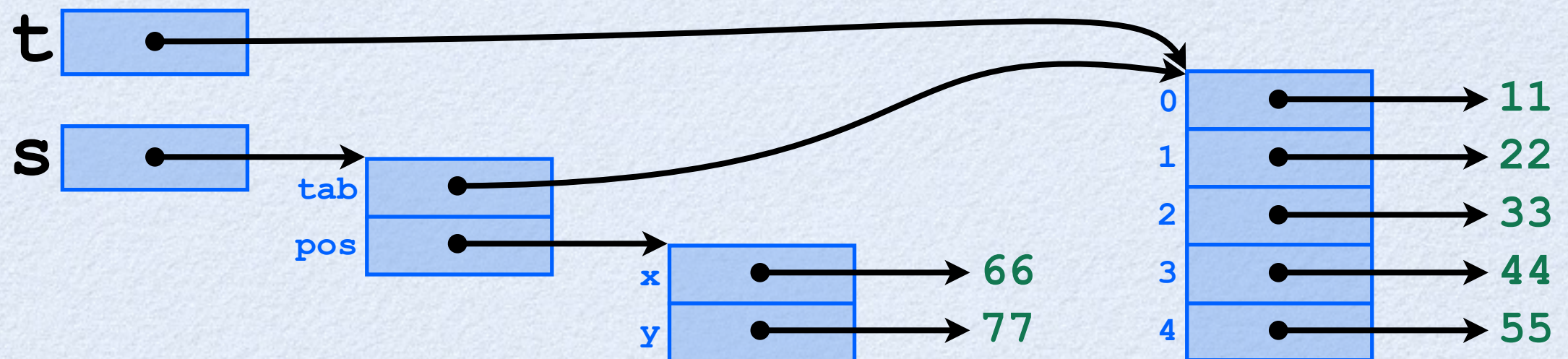
temp[5] = -24; // température du 6 janvier
print( temp[5] ); // imprime -24
```



# Modèle mémoire

- La représentation des structures et tableaux est basée sur le concept de **référence**
- Les cellules mémoire (variables, éléments de tableau, et champs de structure) contiennent des **pointeurs** vers les données contenues

```
> var t = [11, 22, 33, 44, 55];  
> var s = {tab: t, pos: {x: 66, y: 77}};
```



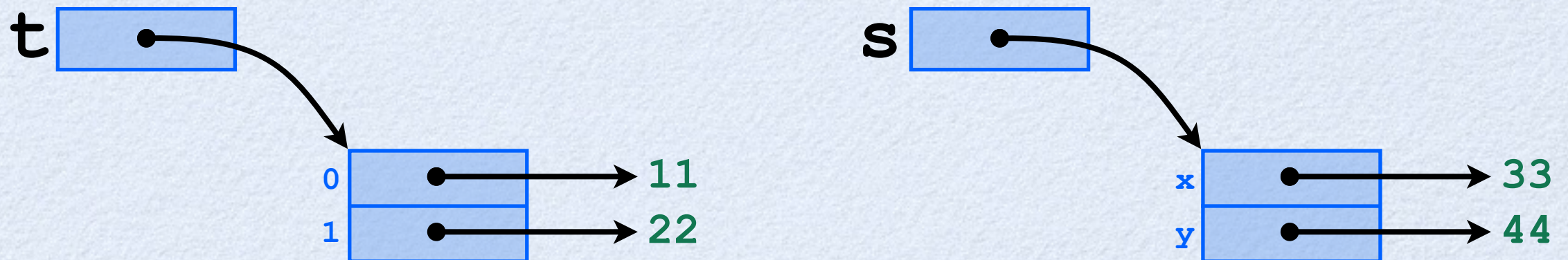


# Modèle mémoire

- Lors d'un appel de fonction, ce sont des **références** qui sont passées en paramètre et retournées comme résultat :

point  
d'exécution

```
var f = function (a, b) { a[0] = b; };  
var t = [11, 22];  
var s = {x: 33, y: 44};  
f(t, s);  
s.x = 55;  
print(t[0].x); // imprime 55
```



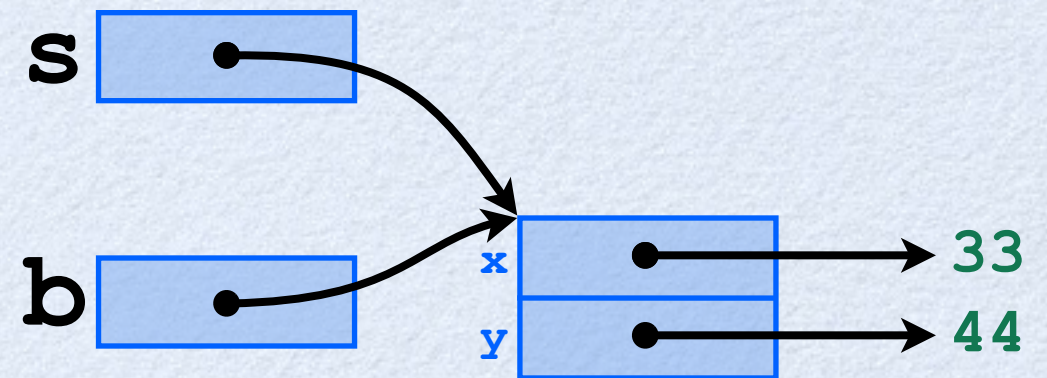
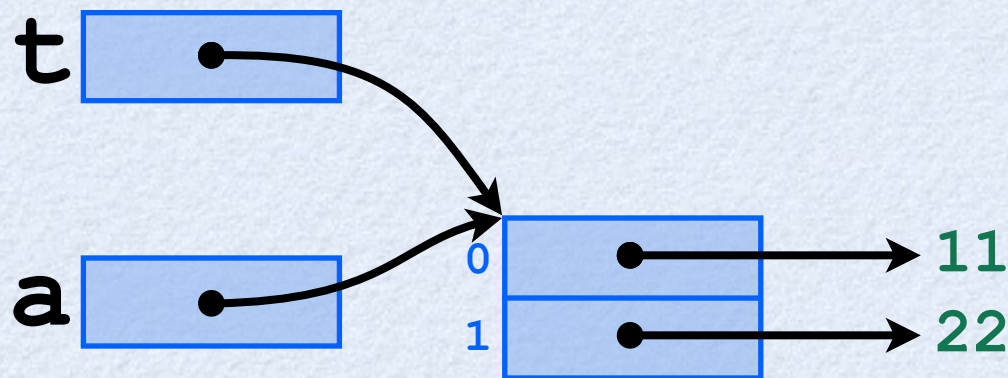


# Modèle mémoire

- Lors d'un appel de fonction, ce sont des **références** qui sont passées en paramètre et retournées comme résultat :

```
var f = function (a, b) { a[0] = b; };  
var t = [11, 22];  
var s = {x: 33, y: 44};  
f(t, s);  
s.x = 55;  
print(t[0].x); // imprime 55
```

point  
d'exécution



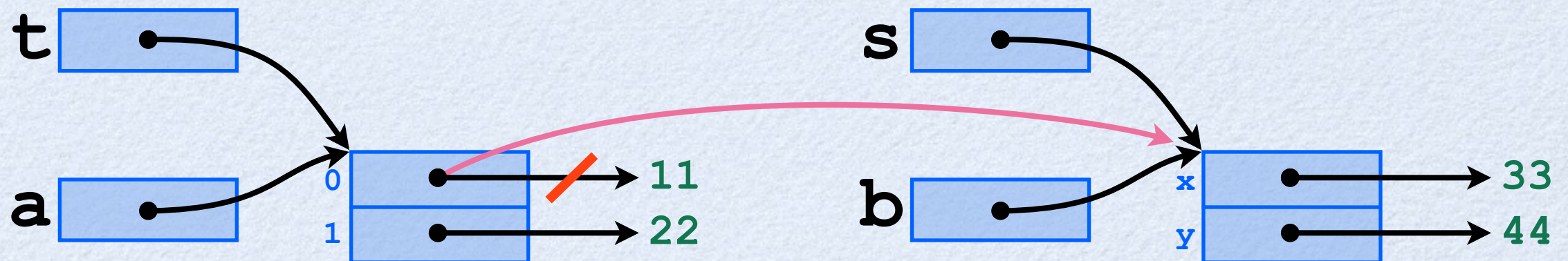


# Modèle mémoire

- Lors d'un appel de fonction, ce sont des **références** qui sont passées en paramètre et retournées comme résultat :

```
var f = function (a, b) { a[0] = b; };  
var t = [11, 22];  
var s = {x: 33, y: 44};  
f(t, s);  
s.x = 55;  
print(t[0].x); // imprime 55
```

point  
d'exécution



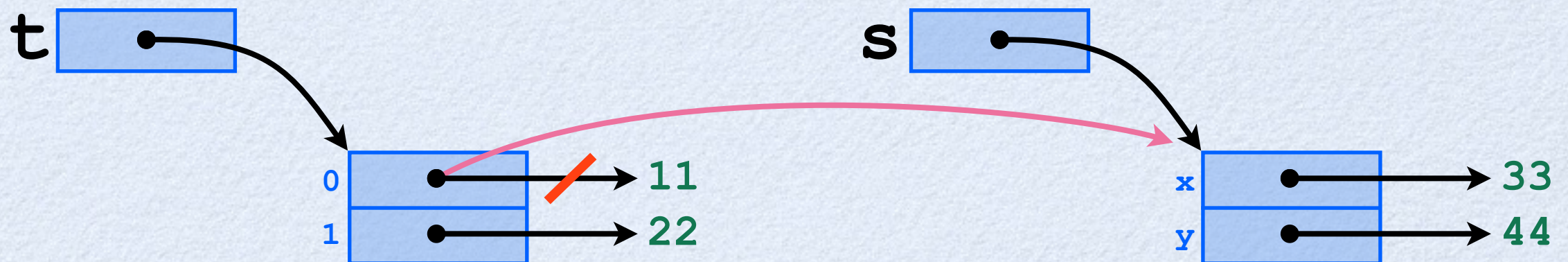


# Modèle mémoire

- Lors d'un appel de fonction, ce sont des **références** qui sont passées en paramètre et retournées comme résultat :

point d'exécution →

```
var f = function (a, b) { a[0] = b; };  
var t = [11, 22];  
var s = {x: 33, y: 44};  
f(t, s);  
s.x = 55;  
print(t[0].x); // imprime 55
```



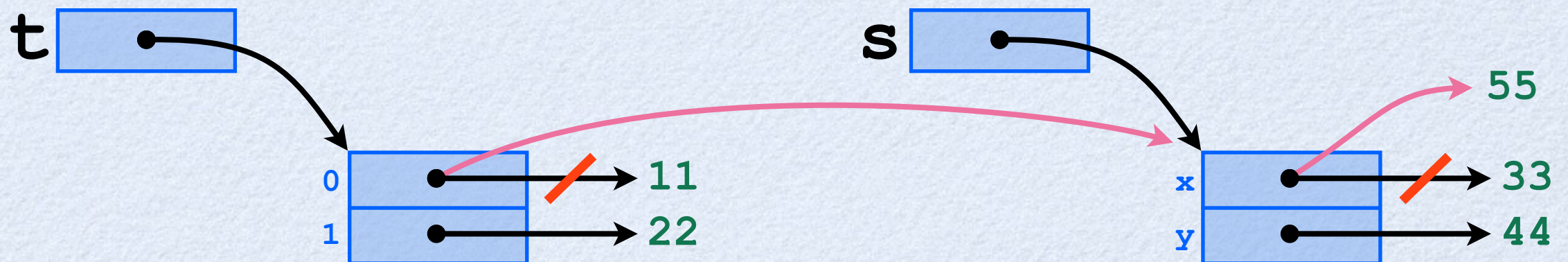


# Modèle mémoire

- Lors d'un appel de fonction, ce sont des **références** qui sont passées en paramètre et retournées comme résultat :

```
var f = function (a, b) { a[0] = b; };  
var t = [11, 22];  
var s = {x: 33, y: 44};  
f(t, s);  
s.x = 55;  
print(t[0].x); // imprime 55
```

point  
d'exécution

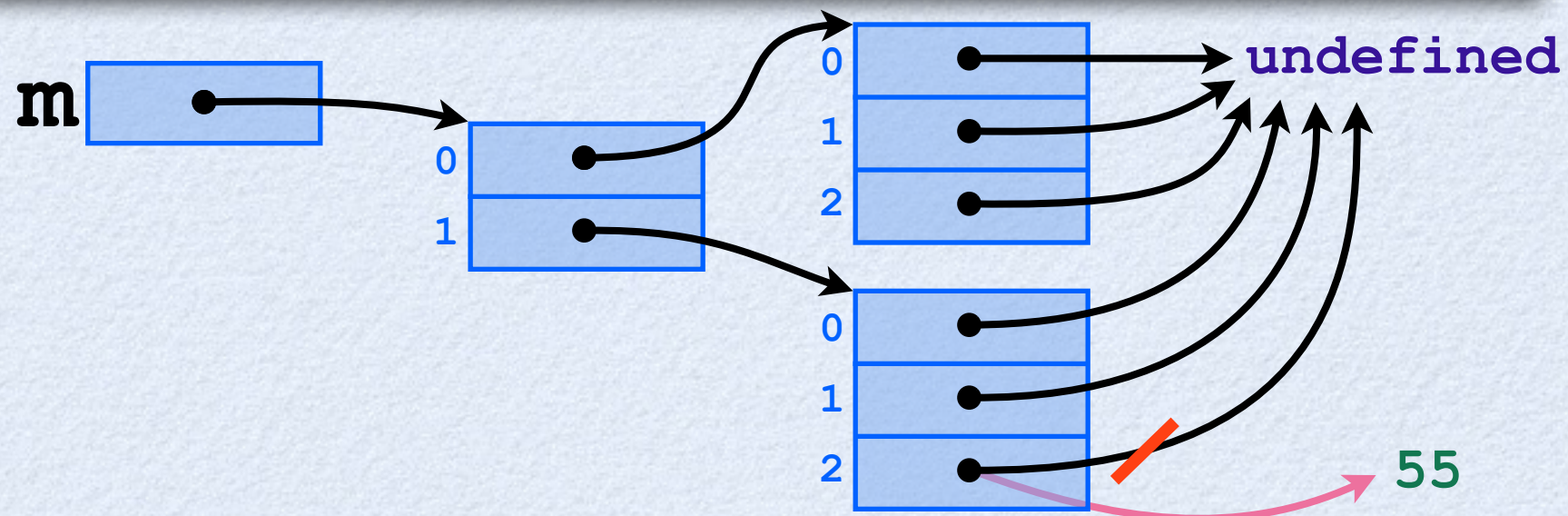




# Modèle mémoire

- Une matrice (grille 2D) se représente comme un tableau de tableaux

```
var creerMatrice = function (nbRangees, nbColonnes) {  
  var resultat = Array(nbRangees);  
  for (var i=0; i<nbRangees; i++) {  
    resultat[i] = Array(nbColonnes);  
  }  
  return resultat;  
};  
  
var m = creerMatrice(2, 3);  
  
m[1][2] = 55;
```

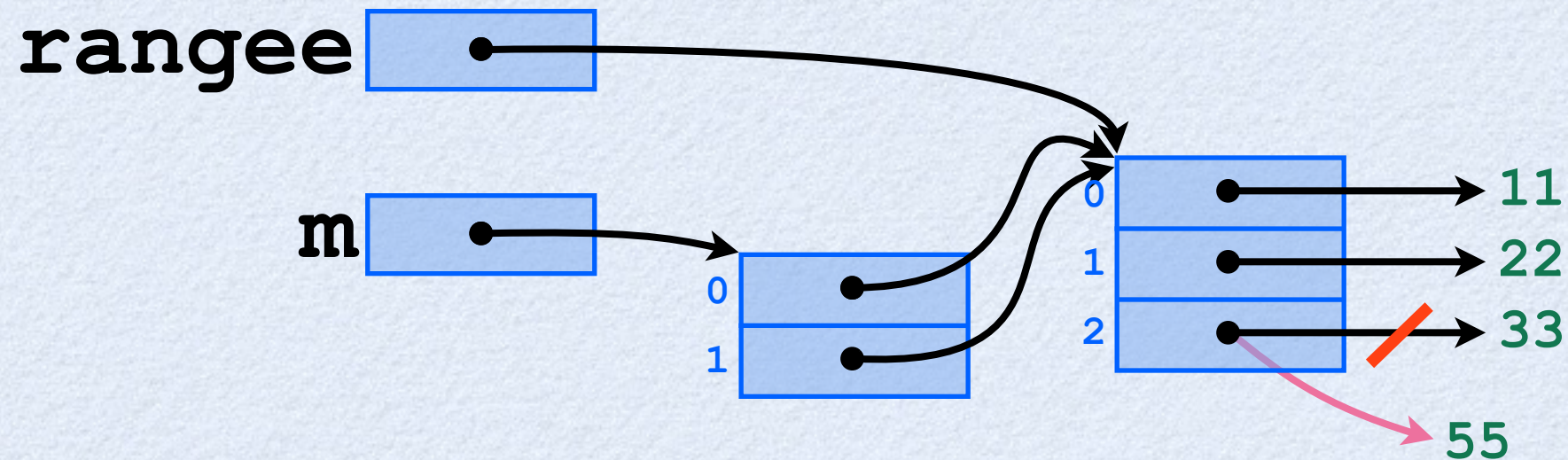




# Modèle mémoire

- Les rangées doivent être **indépendantes**; le code suivant crée une matrice incorrectement :

```
var rangee = [11, 22, 33];  
  
var m = [rangee, rangee]; // rangées partagées!  
  
m[1][2] = 55;  
  
print(m[0][2]); // imprime 55
```





# Traitements de tableaux

- Quelques traitements typiques sur les tableaux :
  - imprimer tous les éléments d'un tableau
  - renverser le contenu d'un tableau
  - trouver la position d'une valeur dans un tableau
  - imprimer la valeur minimale d'un tableau
  - trier un tableau par ordre croissant



// Procédure qui imprime tous les éléments d'un tableau

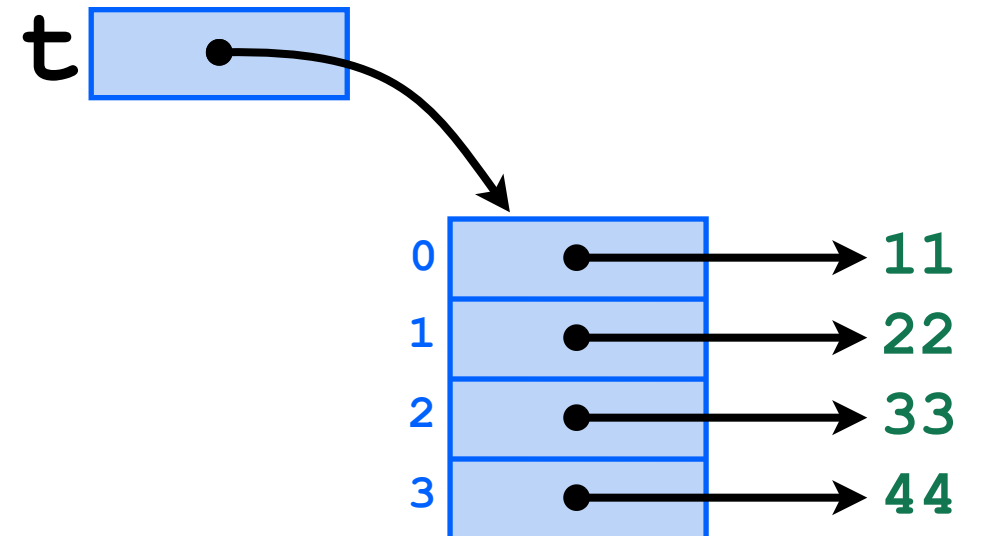
```
var imprimerTab = function (t) {  
    for (var i=0; i<t.length; i++) {  
        print(t[i]);  
    }  
};
```

```
var tab1 = [11, 22, 33, 44];
```

```
var tab2 = ["poire", "orange", "pomme"];
```

```
imprimerTab(tab1);
```

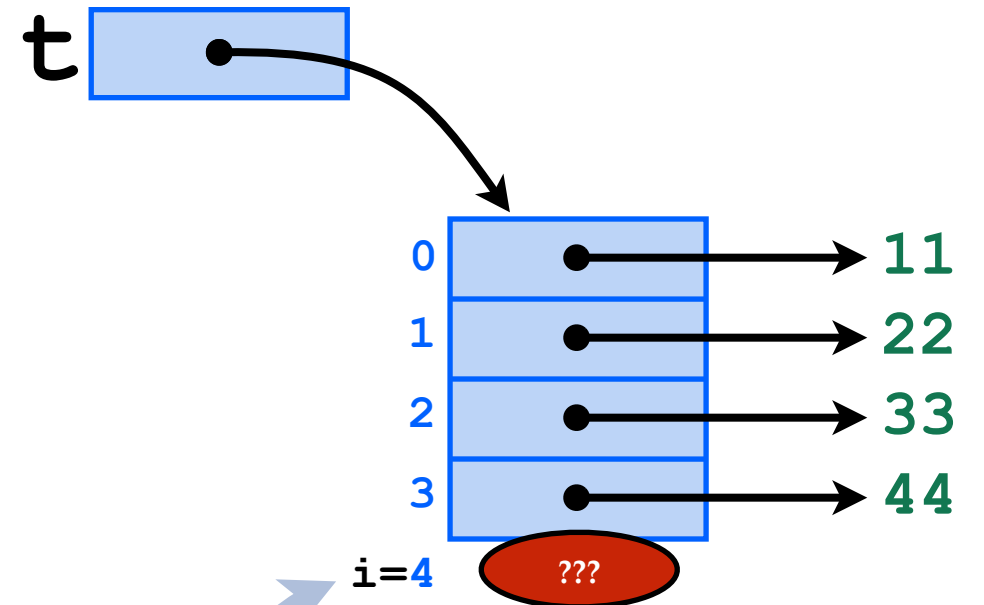
```
imprimerTab(tab2);
```





// Procédure qui imprime tous les éléments d'un tableau

```
var imprimerTab = function (t) {  
  for (var i=0; i<=t.length; i++) {  
    print(t[i]);  
  }  
};  
  
var tab1 = [11, 22, 33, 44];  
var tab2 = ["poire", "orange", "pomme"];  
  
imprimerTab(tab1);  
imprimerTab(tab2);
```

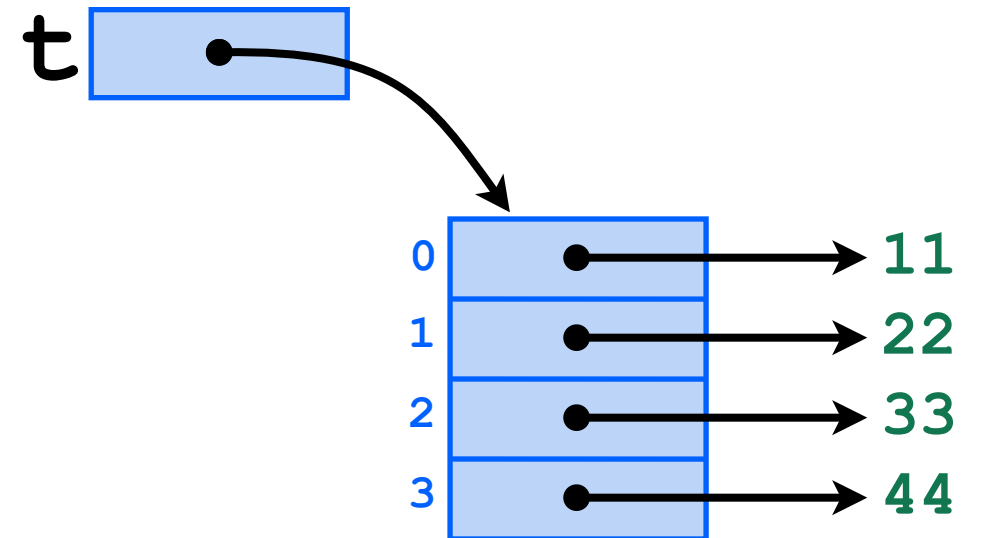


attention aux erreurs "off by one"  
*array index is out of bounds*



// Procédure qui renverse le contenu d'un tableau in-situ

```
var renverserTab = function (t) {  
    for (var i=0; i<t.length/2; i++) {  
        var j = t.length-i-1;  
        var temp = t[i];  
        t[i] = t[j];  
        t[j] = temp;  
    }  
};
```



```
var tab1 = [11, 22, 33, 44];
```

```
var tab2 = ["poire", "orange", "pomme"];
```

```
renverserTab(tab1); print(tab1); // imprime : 44,33,22,11
```

```
renverserTab(tab2); print(tab2); // imprime : pomme,orange,poire
```



// Procédure qui renverse le contenu d'un tableau in-situ

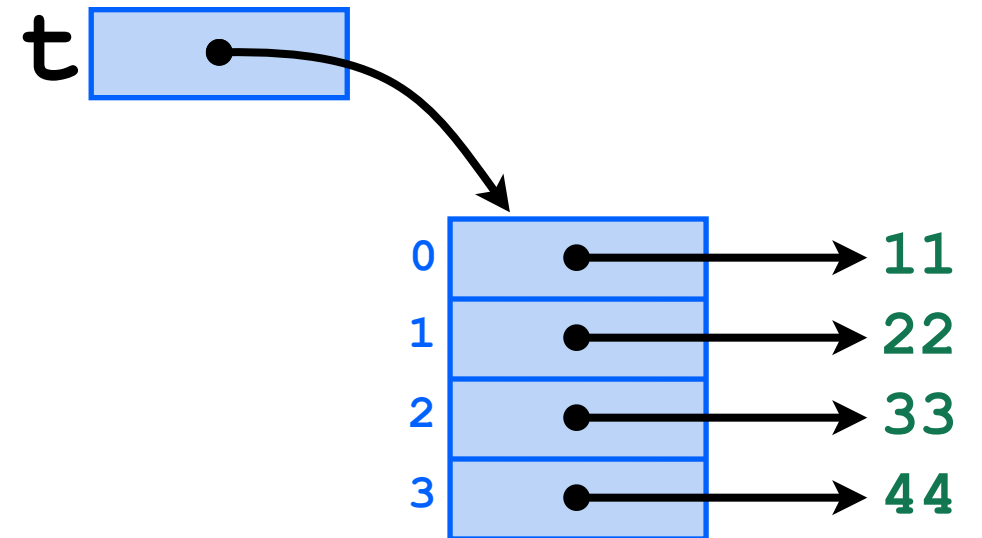
```
var renverserTab = function (t) {  
  
    var i = 0;  
    var j = t.length - 1;  
  
    while (i < j) {  
        var temp = t[i];  
        t[i] = t[j];  
        t[j] = temp;  
        i++;  
        j--;  
    }  
  
};
```

```
var tab1 = [11, 22, 33, 44];
```

```
var tab2 = ["poire", "orange", "pomme"];
```

```
renverserTab(tab1);    print(tab1); // imprime : 44,33,22,11
```

```
renverserTab(tab2);    print(tab2); // imprime : pomme,orange,poire
```





// Fonction qui renverse le contenu d'un tableau

```
var renverserTab = function (t) {  
    var resultat = Array(t.length);  
    for (var i=0; i<t.length; i++) {  
        resultat[i] = t[t.length-i-1];  
    }  
}
```

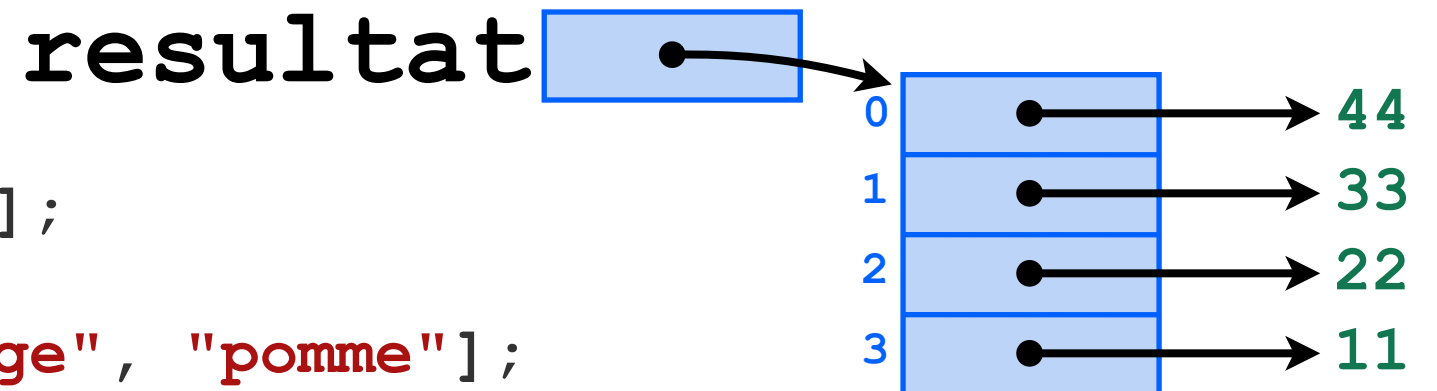
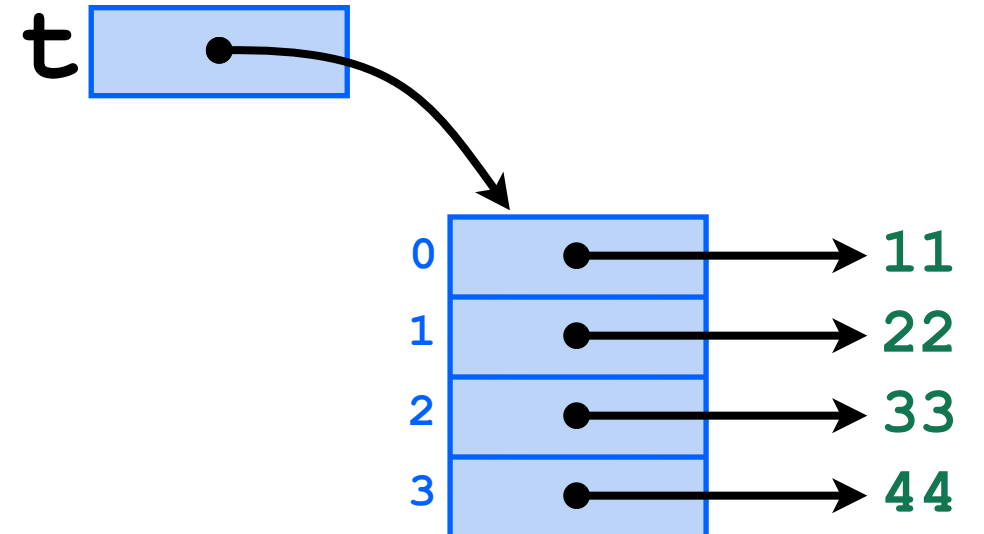
```
    return resultat;  
};
```

```
var tab1 = [11, 22, 33, 44];
```

```
var tab2 = ["poire", "orange", "pomme"];
```

```
print(renverserTab(tab1)); // imprime : 44,33,22,11
```

```
print(renverserTab(tab2)); // imprime : pomme,orange,poire
```





// Fonction qui trouve la position d'une valeur dans un tableau

```
var position = function (t, val) {  
    for (var i=0; i<t.length; i++) {  
        if (t[i] == val) {  
            return i; // on a trouvé!  
        }  
    }  
  
    return -1; // code indiquant échec  
};
```

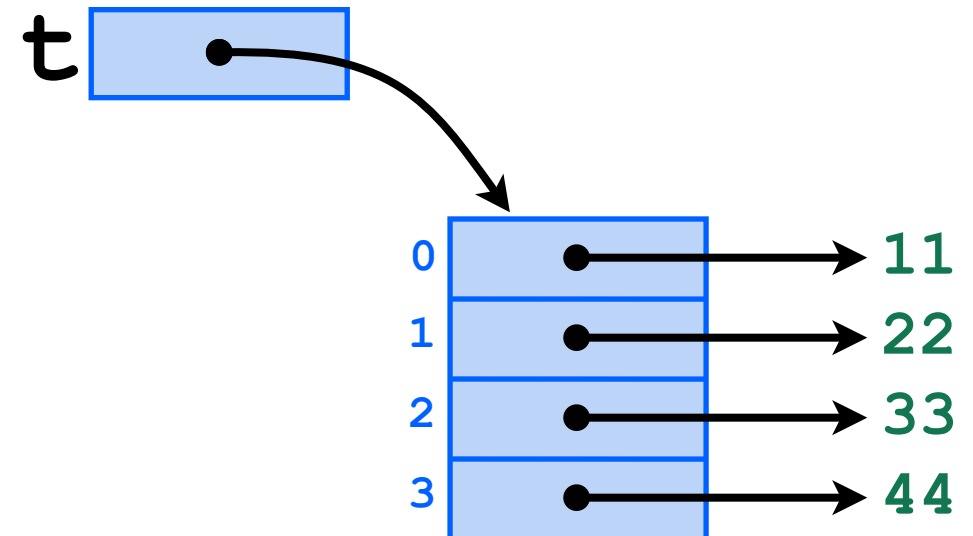
```
var tab1 = [11, 22, 33, 44];
```

```
var tab2 = ["poire", "orange", "pomme"];
```

```
print(position(tab1, 22)); // imprime : 1
```

```
print(position(tab1, 100)); // imprime : -1
```

```
print(position(tab2, "pomme")); // imprime : 2
```





```

// Fonction qui retourne la valeur minimale d'un tableau

var minimum = function (t) {

    // suppose que t.length >= 1

    var min = t[0];

    for (var i=1; i<t.length; i++) {
        if (t[i] < min) {
            min = t[i];
        }
    }

    return min;
};

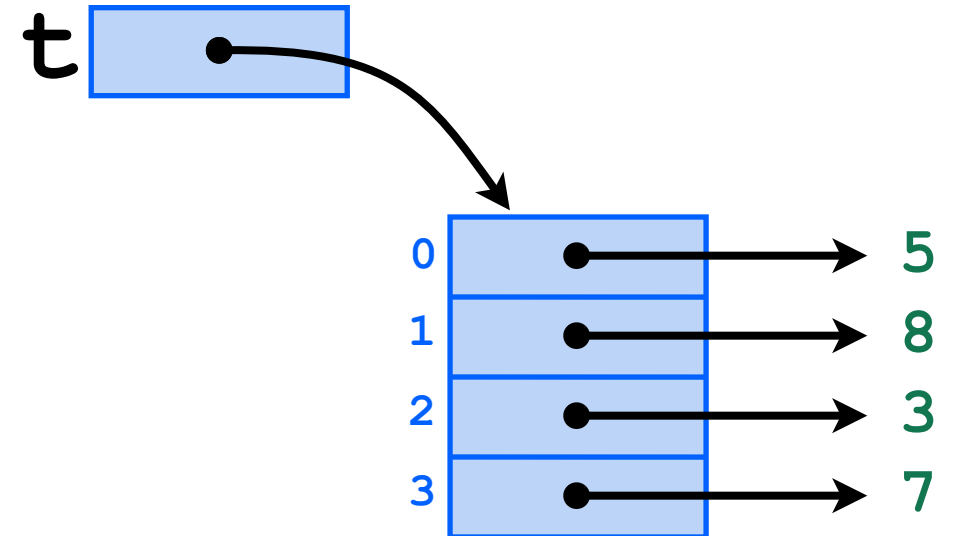
var tab1 = [5, 8, 3, 7];

var tab2 = ["poire", "orange", "pomme"];

print(minimum(tab1)); // imprime : 3

print(minimum(tab2)); // imprime : orange

```



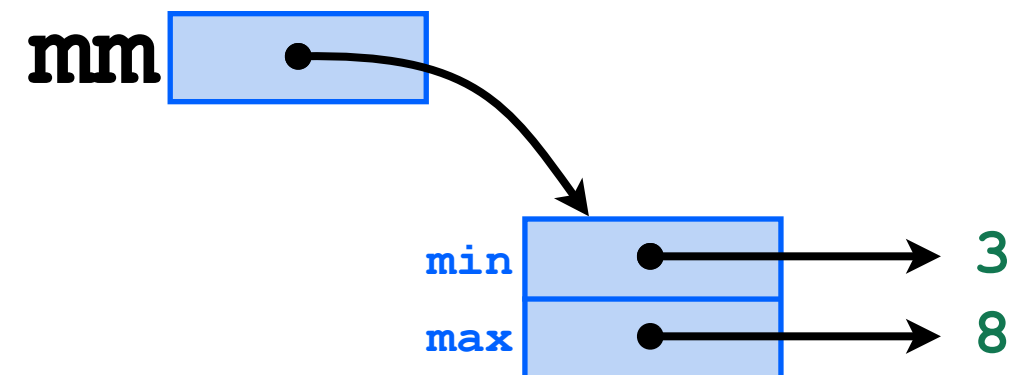
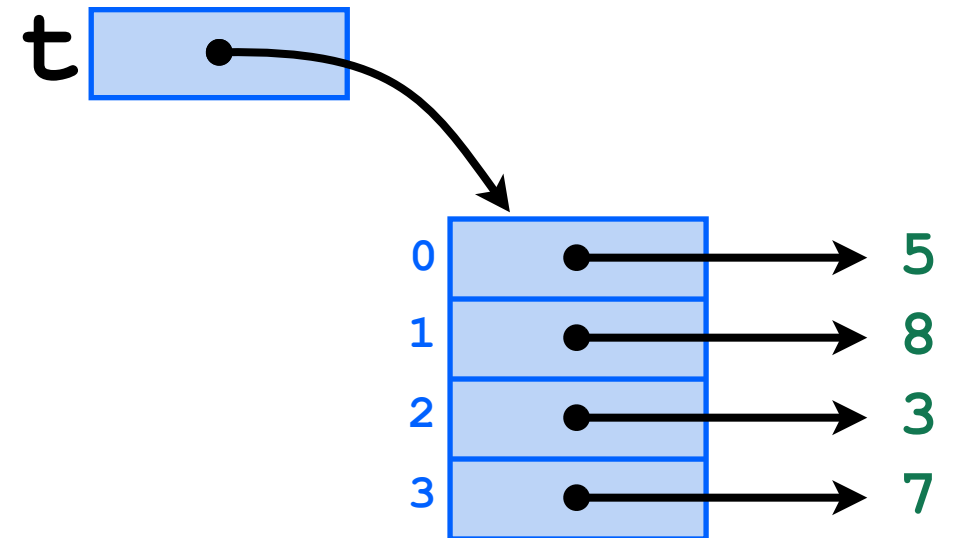


```
// Fonction qui retourne la valeur minimale et la valeur  
// maximale d'un tableau
```

```
var minmax = function (t) {  
    // suppose que t.length >= 1  
  
    var mm = { min: t[0], max: t[0] };  
  
    for (var i=1; i<t.length; i++) {  
        if (t[i] < mm.min) {  
            mm.min = t[i];  
        } else if (t[i] > mm.max) {  
            mm.max = t[i];  
        }  
    }  
  
    return mm;  
};
```

```
var tab = [5, 8, 3, 7];  
var res = minmax(tab);
```

```
print(res.min); // imprime : 3  
print(res.max); // imprime : 8
```

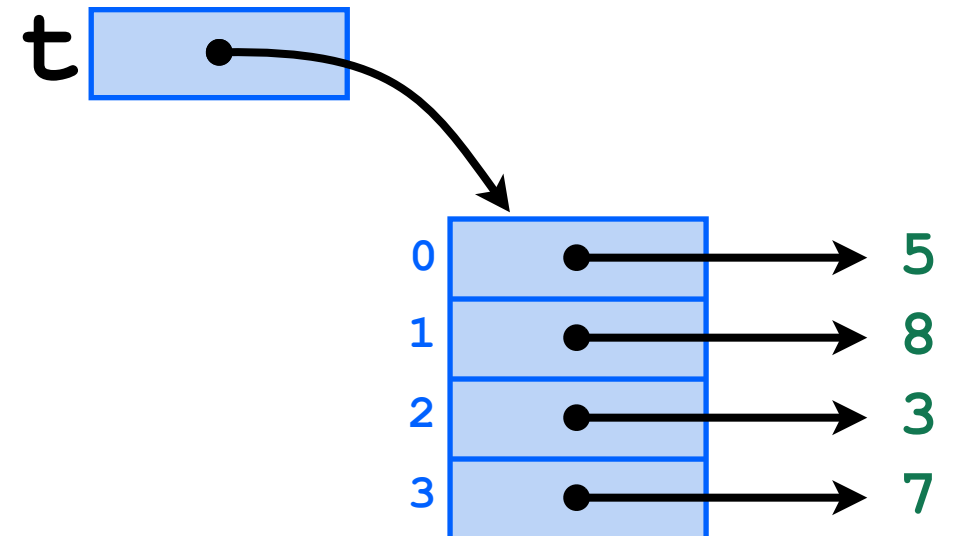




```
// Fonction qui retourne la valeur minimale d'un tableau
```

```
var minimum = function (t) {  
    return t[ positionMin(t) ];  
};  
  
var positionMin = function (t) {  
    // suppose que t.length >= 1  
  
    var posMin = 0;  
  
    for (var i=1; i<t.length; i++) {  
        if (t[i] < t[posMin]) {  
            posMin = i;  
        }  
    }  
  
    return posMin;  
};
```

```
var tab1 = [5, 8, 3, 7];  
var tab2 = ["poire", "orange", "pomme"];  
  
print(minimum(tab1)); // imprime : 3  
print(minimum(tab2)); // imprime : orange
```





// Procédure qui trie un tableau en ordre croissant in-situ

```
var trier = function (t) { // tri par sélection
  for (var i=0; i<t.length-1; i++) {
    var m = positionMin(t, i);
    var temp = t[i];
    t[i] = t[m];
    t[m] = temp;
  }
};
```

```
var positionMin = function (t, debut) {
  // suppose que t.length > debut

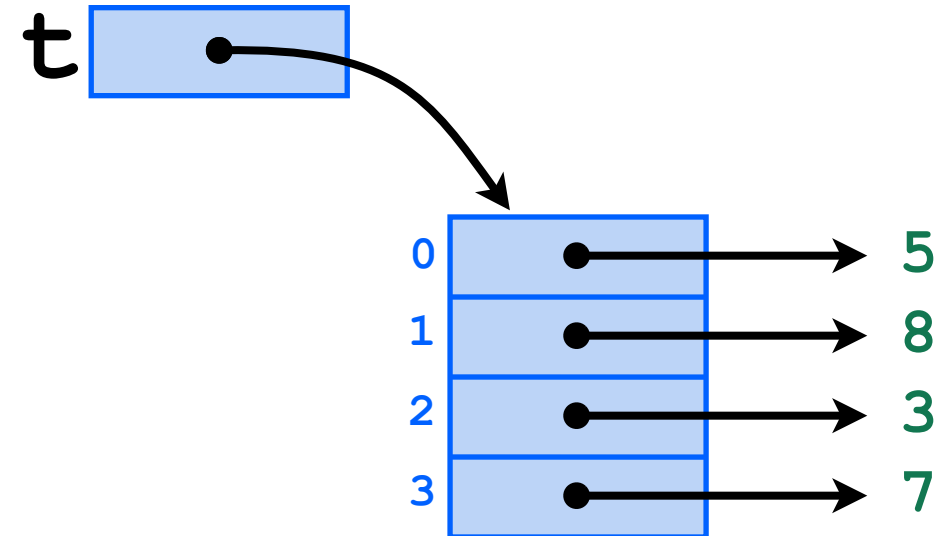
  var posMin = debut;

  for (var i=debut+1; i<t.length; i++) {
    if (t[i] < t[posMin]) {
      posMin = i;
    }
  }

  return posMin;
};
```

```
var tab1 = [5, 8, 3, 7];
var tab2 = ["poire", "orange", "pomme"];
```

```
trier(tab1); print(tab1); // imprime : 3,5,7,8
trier(tab2); print(tab2); // imprime : orange,poire,pomme
```





```
// Procédure qui trie un tableau en ordre croissant in-situ
```

```
var trier = function (t) { // tri bulle

    var echange;

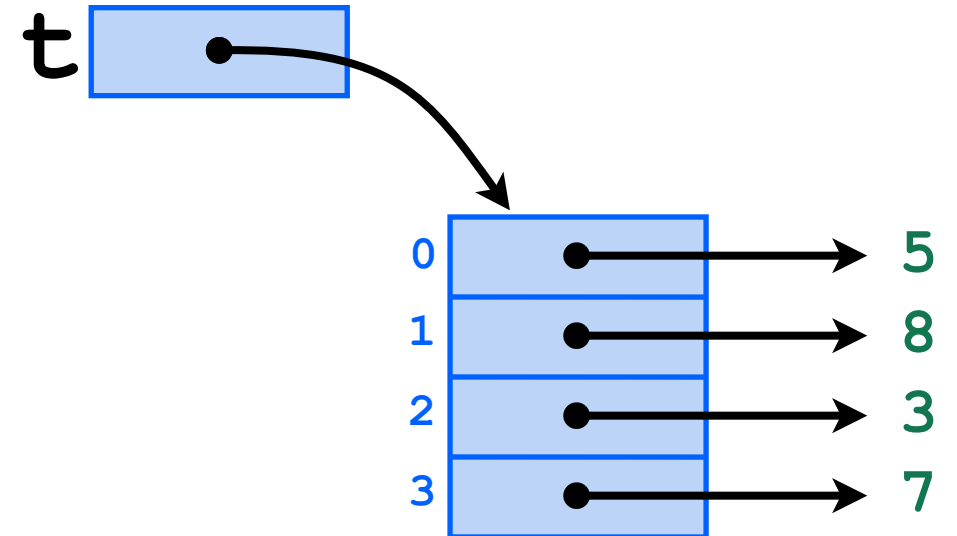
    do {
        echange = false;

        for (var i=0; i<t.length-1; i++) {
            if (t[i+1] < t[i]) {
                var temp = t[i];
                t[i] = t[i+1];
                t[i+1] = temp;
                echange = true;
            }
        }
    } while (echange);

};
```

```
var tab1 = [5, 8, 3, 7];
var tab2 = ["poire", "orange", "pomme"];
```

```
trier(tab1);    print(tab1);    // imprime : 3,5,7,8
trier(tab2);    print(tab2);    // imprime : orange,poire,pomme
```





# Méthodes des types prédéfinis



# Méthodes des types prédéfinis

- JS a un ensemble d'**opérations prédéfinies**
- Certaines opérations, comme l'addition et la multiplication, sont exposées au programmeur sous forme d'**opérateurs** du langage (tels **+** et **\***)
- D'autres opérations sont exposées sous forme de **méthodes**
- Syntaxe :  $\langle exp. \rangle \cdot \langle id. \rangle (\langle exp. \rangle, \dots)$

donnée sur laquelle  
s'applique la méthode  
(objet **receveur**)

nom de la  
méthode

paramètres de  
la méthode



# Méthodes importantes sur les tableaux

- **`t.length`**      (*méthode spéciale sans paramètres*)
  - retourne le nombre d'éléments dans le tableau *t*
- **`t.push(x)`**
  - Ajoute l'élément *x* à la fin du tableau *t* (dont la longueur est donc augmentée de 1) et cette nouvelle longueur est retournée par **push**
- **`t.pop()`**
  - Retire le dernier élément du tableau *t* et le retourne (la longueur de *t* sera donc réduite de 1)



# Méthodes importantes sur les tableaux

- ***t1*.concat(*t2*)**
  - retourne un **nouveau tableau** qui contient tous les éléments de *t1* suivis des éléments de *t2*
- ***t*.slice(*i*,*j*)**
  - retourne un **nouveau tableau** qui contient les éléments de *t* entre l'index *i* (inclus) et *j* (exclus)



# Example 1

```
var t = [11,22];
```

```
t.push(33);
```

```
t.push(44);
```

```
t.push(567);
```

```
print(t);
```

11, 22, 33, 44, 567

```
var x = t.pop();
```

```
print(t);
```

```
print(x);
```

11, 22, 33, 44  
567

```
t.push(55);
```

```
print(t);
```

11, 22, 33, 44, 55

```
print( t.concat([66,77]) );
```

11, 22, 33, 44, 55, 66, 77

```
print(t);
```

11, 22, 33, 44, 55

```
print( t.slice(1,3) );
```

22, 33



# Exemple 2

```
var extrairePositifs = function (tableau) {  
  
    var n = 0;  
  
    for (var i=0; i<tableau.length; i++) {  
        if (tableau[i] >= 0) {  
            n++;  
        }  
    }  
  
    var resultat = Array(n);  
    var j = 0;  
  
    for (var i=0; i<tableau.length; i++) {  
        if (tableau[i] >= 0) {  
            resultat[j++] = tableau[i];  
        }  
    }  
  
    return resultat;  
};  
  
var tab = [3, -40, 5, 1, -25, 7];  
  
print( extrairePositifs(tab) ); // imprime 3,5,1,7
```



# Exemple 2 avec push

```
var extrairePositifs = function (tableau) {  
    var resultat = [];  
  
    for (var i=0; i<tableau.length; i++) {  
        if (tableau[i] >= 0) {  
            resultat.push(tableau[i]);  
        }  
    }  
  
    return resultat;  
};  
  
var tab = [3, -40, 5, 1, -25, 7];  
  
print( extrairePositifs(tab) ); // imprime 3,5,1,7
```



# Question intra

```
var retirerSequencesEgales = function (tab) {  
    var resultat = [];  
    if (tab.length > 0) { // doit avoir au moins 1 élément  
        resultat.push(tab[0]);  
        for (var i=1; i<tab.length; i++) {  
            if (tab[i] != tab[i-1]) { // séquence inégale?  
                resultat.push(tab[i]);  
            }  
        }  
    }  
    return resultat;  
};  
  
var testRetirerSequencesEgales = function () {  
    assert( retirerSequencesEgales([]) == '' );  
    assert( retirerSequencesEgales([1]) == '1' );  
    assert( retirerSequencesEgales([1,2,2,2,3]) == '1,2,3' );  
    assert( retirerSequencesEgales([1,1,8,1,1]) == '1,8,1' );  
    assert( retirerSequencesEgales([5,5,5,5,5]) == '5' );  
};  
  
testRetirerSequencesEgales();  
  
print(retirerSequencesEgales(["je", "ne", "ne", "begaie", "plus", "plus"]));
```