IFT1015 Programmation 1

Données structureés

(structures et tableaux)

Matériel produit par Marc Feeley

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 2011
  - 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**)
- Normalement, 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)
  - …

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 2011 (sur les 365 éléments, on pourrait être intéressé au 5ieme, la température du 5 janvier 2011, ou au 365ieme, la température du 31 décembre 2011)
- 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**
- L'accès à un champ se fait avec la syntaxe :
  ‹*expression*› **.** ‹*identificateur*›

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

var coord = { 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
```

Tableaux

- Un tableau peut être construit en énumérant tous ces éléments entre « **[** » et « **]** »
- L'accès à un élément, l'**indexation**, se fait avec la syntaxe : *‹expression›* **[***‹expression›***]**

```
var couleurs = ["rouge", "vert", "bleu"];

var lotto649 = [34, 10, 23, 1, 49, 22];

var noel = [2012, 12, 25];

print( couleurs[1] ); // imprime vert
couleurs[1] = "jaune";
print( couleurs[1] ); // imprime jaune
```

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 :

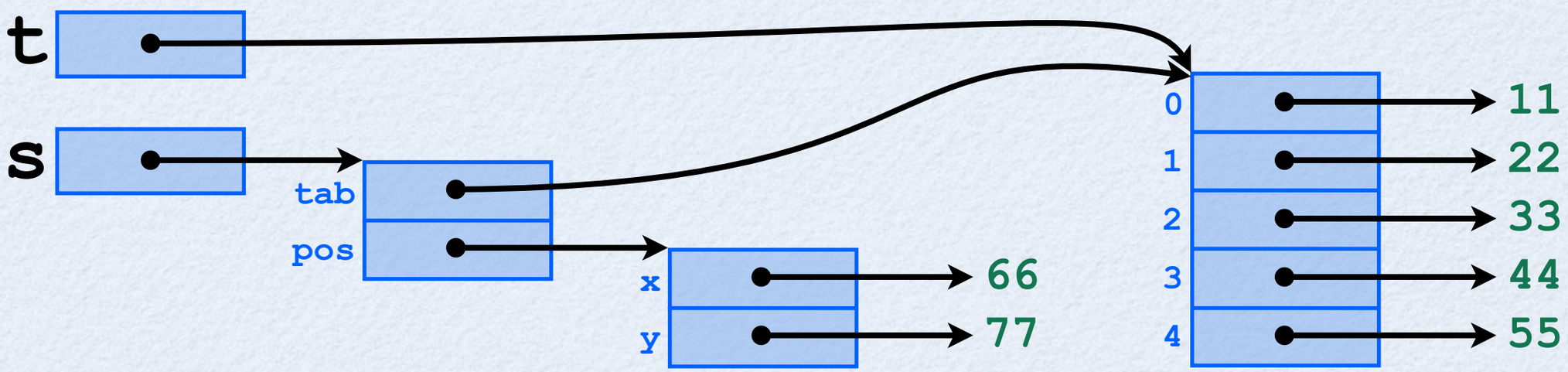
‹*expression*› **`.length`**

```
var temp = Array(365);  // températures de 2011

print( temp.length ); // imprime 365

temp[5] = -15;
print( temp[5] ); // imprime -15
```

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

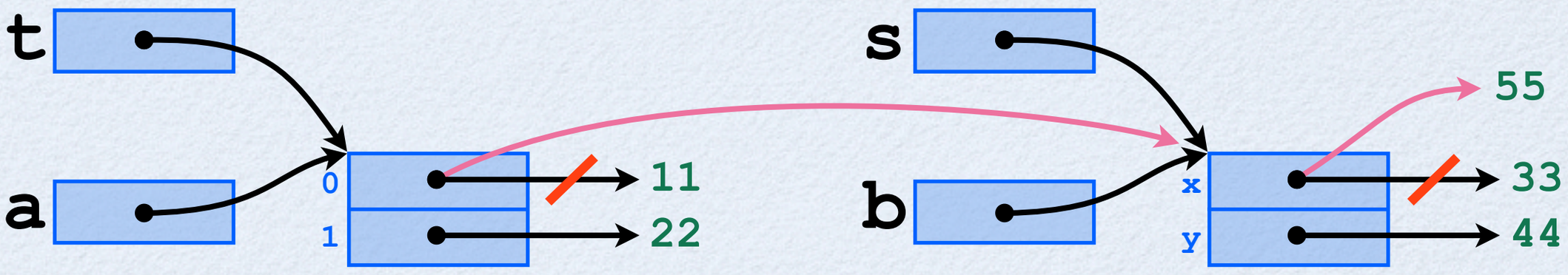
```
var t = [11, 22, 33, 44, 55];
var s = {tab: t, pos: {x: 66, y: 77}};
```



Modèle mémoire

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

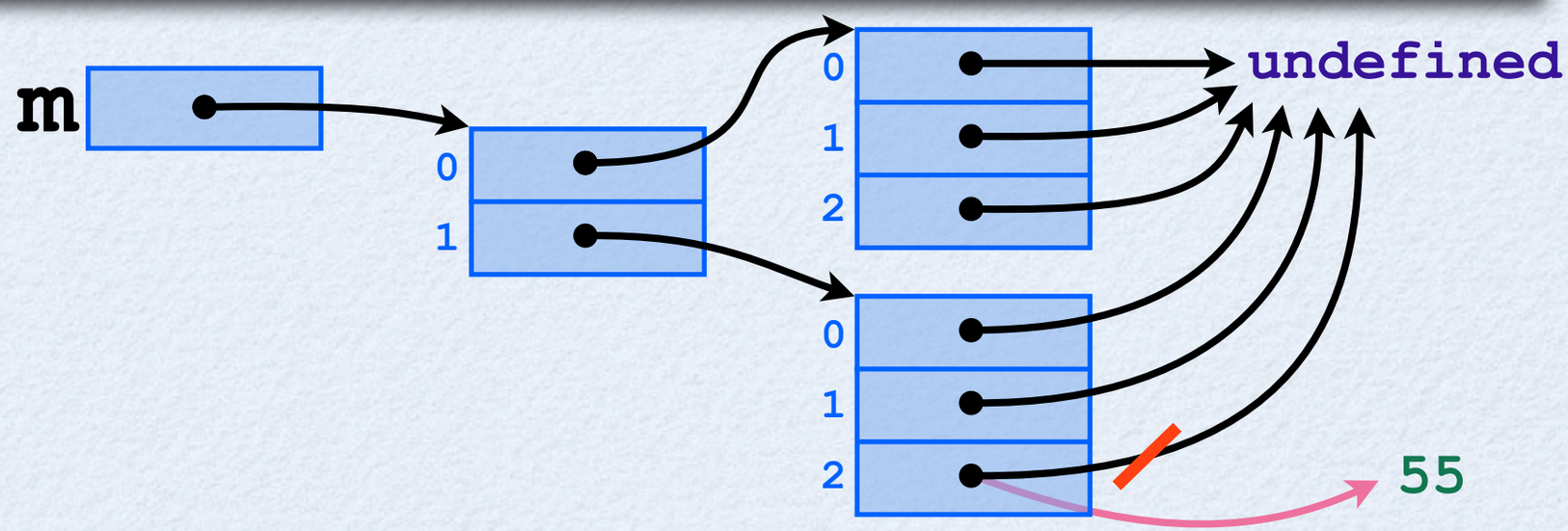
```
var f = function (a, b) { a[0] = b; };
var t = [11, 22];
var s = {x: 33, y: 44};
f(t, s);
s.x = 55;
print(t[0].x); // imprime 55
```



Modèle mémoire

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

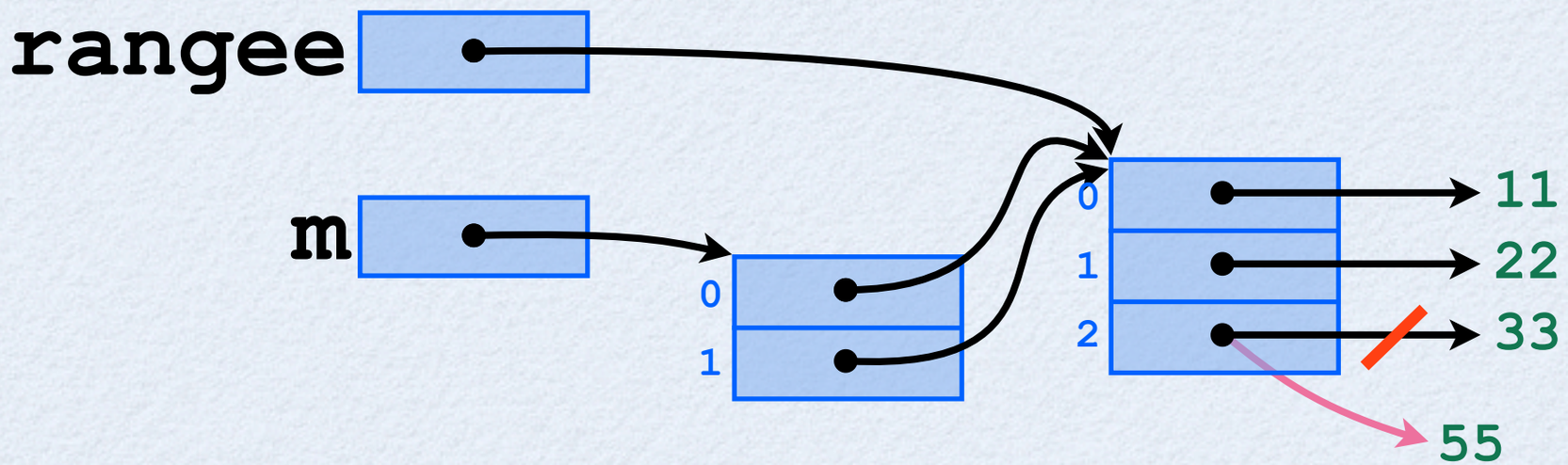
```
var creerMatrice = function (nbRangees, nbColonnes) {
    var resultat = Array(nbRangees);
    for (var i=0; i<nbRangees; i++) {
        resultat[i] = Array(nbColonnes);
    }
    return resultat;
};

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 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 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

    do {
        var 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 (tel **+** et *****)
- D'autres opérations sont exposées sous forme de **méthodes**
- Syntaxe : *‹exp.›* **.** *‹id.›* **(** *‹exp.›* **,** … **)**

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

nom de la méthode

paramètres de la méthode

Méthodes sur les tableaux
(les plus importantes)

- *t*`.length`
  longueur du tableau (pas de parenthèses)
- *t*`.push(`*x*`)`
  - Ajoute l'élément *x* à la fin du tableau
    La longueur de *t* sera donc augmentée de 1.
    Cette nouvelle longueur est retournée
- $x = t$`.pop()`
  - Enlève le dernier élément du tableau *t* et le retourne.
    La longueur de *t* sera donc diminuée de 1.

Méthodes sur les tableaux
(les plus importantes)

- *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 une **copie** des éléments de *t* entre l'index *i* (inclus) et *j* (exclus)

Exemple 1

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

t.push(33);
t.push(44);
t.push(567);
print(t);  // imprime 11,22,33,44,567

var x = t.pop();
print(t);  // imprime 11,22,33,44
print(x);  // imprime 567

t.push(55);
print(t);  // imprime 11,22,33,44,55

print( t.concat([66,77]) ); // imprime 11,22,33,44,55,66,77

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

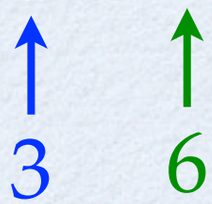
Méthodes sur les string
(les plus importantes)

- *t* . **`length`** *(méthode spéciale sans paramètres)*
  - retourne le nombre de caractères dans le texte *t*
- *t* . **`charAt(`***i***`)`**
  - retourne un **nouveau texte** de longueur 1, le caractère à l'index *i* du texte *t*
- *t* . **`charCodeAt(`***i***`)`**
  - retourne un entier qui est le **code Unicode** du caractère à l'index *i* du texte *t*
- ***t*.substring(pos_debut, pos_fin_exclue)**

  retourne une sous-chaîne

Extraction de sous-chaînes

- Ex: soit **s** une expression de type string
- **s.charAt(i)** retourne le caractère à la position i
  les caractères sont numérotés de 0 à longueur-1
  Ex: **"Hello World!".charAt(0)** vaut **"H "**
  "Hello World!".charAt(11) vaut **"! "**
- **s.substring(pos_debut, pos_fin_exclue)**
  retourne une sous-chaîne
  Ex: **"Bonjour".substring(3,6)** vaut **"jou"**



Similarité entre opérations sur string et tableau

```
> var t = "ABC";

> t.length
  3

> t.charAt(1)
  "B"

> t.substring(1,3);
  "BC"

> t+"DEF";
  "ABCDEF"

> t.charCodeAt(0);
  65
```

```
> var t = ["A","B","C"];

> t.length
  3

> t[1]
  "B"

> t.slice(1,3);
  ["B","C"]

> t.concat(["D","E","F"]);
  ["A","B","C","D","E","F"]

> t.concat(["DEF"]);
  ["A","B","C","DEF"]

> t.concat("DEF");
  ["A","B","C","DEF"]
```

Exemple : encodagePositionnel

```
var encodagePositionnel = function (n, base) {

    // Cette fonction retourne l'encodage du nombre n dans la
    // base spécifiée, qui peut être entre 2 et 36.

    var chiffres = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    var e = "";                          // pour accumuler l'encodage

    do {
        e = chiffres.charAt(n % base) + e; // accumuler un chiffre
        n = Math.floor(n / base); // passer aux autres chiffres
    } while (n > 0);                     // tant qu'il reste des chiffres

    return e;
};

print( "0x" + encodagePositionnel(51966, 16) );  // imprime 0xCAFE
```