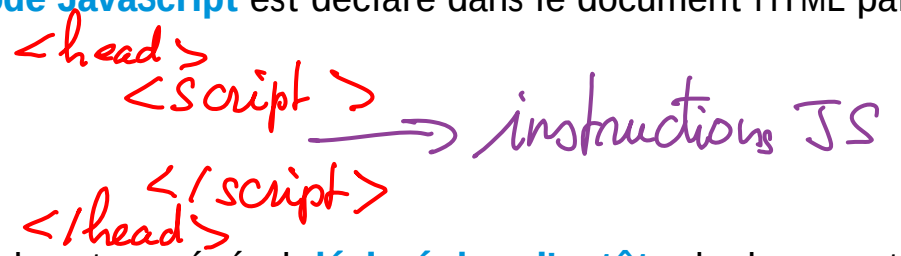
Le **code JavaScript** est déclaré dans le document HTML par les balises HTML <script> et </script>:

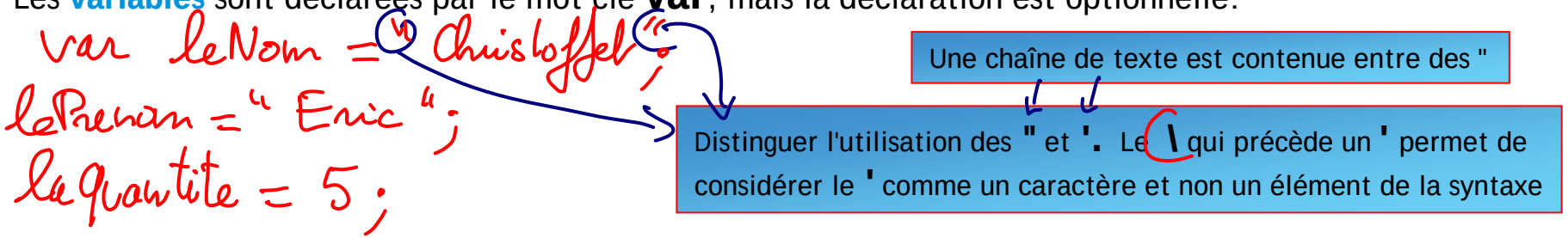


Le code est en général **déclaré dans l'entête** du document, soit entre les balises <head> … </head>.

Plutôt que d'écrire le code JavaScript dans le fichier HTML, il peut être écrit dans un **fichier annexe**, enregistré avec l'extension **.js** , soit le fichier **leCodeJS.js** , **l'attribut SRC** permet d'y faire référence:

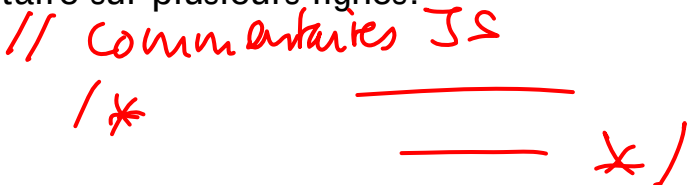
Les **instructions** JavaScript s'écrivent une par ligne, et **se terminent** par un **;**

Les **variables** sont déclarées par le mot clé **var**, mais la déclaration est optionnelle:



→Les **variables ne sont pas typées** (inutile de préciser si c'est un nombre, du texte, un objet…).

Les commentaires sont précédés de **//** pour un commentaire sur 1 seule ligne, ou entre **/*** et ***/** pour un commentaire sur plusieurs lignes:



Les **opérateurs arithmétiques** classiques: **+ , - , * , /, +=, % (modulo)**

```
a=3;
b= a+2; // b → 5
```

L'incrément ou décrément d'une unité: **++ , --**

```
i=i+1;
i++;
```

Les opérateurs agissant sur les chaînes de texte: **+ , +=** , c'est une **concaténation** de chaînes de texte

```
leNom="Christoffel";
lePrenom="Eric";
lIdentite = leNom+" "+lePrenom // déclaration d'une variable contenant du texte
                               //concaténation de chaîne de texte et de variables
↳ ChristoffelEric
// concaténation de chaînes
// " la suite du message" est rajoutée au contenu de texte1
```

Les opérateurs de **comparaison** utilisés dans les test conditionnels **if** : **== , != , > , < , >= , <=**

```
if (a==b) { instruction; }
```

↑ égalité entre 2 nombres

Les **opérateurs logiques** permettent d'effectuer plusieurs comparaisons : **&& , || , !**

```
if( a==b && a>2)   // opérateur AND (ET), les 2 comparaisons doivent être satisfaites
if (a==2 || a==3)  // opérateur OR (OU), l'une ou l'autre comparaison doit être satisfaite
```

↖ Alt Gr + 6

// opérateur NOT (non), équivalent au contraire, cas des valeurs booléenne, retourne **true** si *a*=**false**

Syntaxe for

```
for (i=0; i<Nmax; i++){
instructions;
instructions;
}
```

Algorithme

Les instructions sont exécutées tant que la condition est vérifiée.

Syntaxe while

```
i < Nmax
while ( condition ){
instructions;
i++;
}
```

Si le test conditionnel est vérifié, alors les instructions sont exécutées. Les instructions peuvent ne jamais être exécutées!

Attention, il faut que la *condition* se modifie dans la boucle, afin que le test conditionnel change d'état à un moment donné!

Syntaxes ≡ PHP

On souhaite calculer la somme des nombres de 1 à *N*.

Schéma d'exécution : ≡ Algorithme

a) Pour cela, déclarez la limite *N*, initialisez une variable *somme* à 0.

b) L'instruction **for** :
(1) initialise une variable *i* à 0 ;
(2) effectue un test conditionnel, à savoir si *i*<=*N* ;
(3) exécute une série d'instructions séquentiellement si le test conditionnel retourne *true* ;
(4) incrémente la variable *i*.

c) Affichez le résultat à l'aide la méthode *window.alert( )*.

```
<script>
    N = 10;
    somme = 0;
    for (i = 0; i <= N; i++){
    //somme = somme + i;
      somme += i;
    }
    window.alert ("La somme totale est : " + somme );
</script>
```

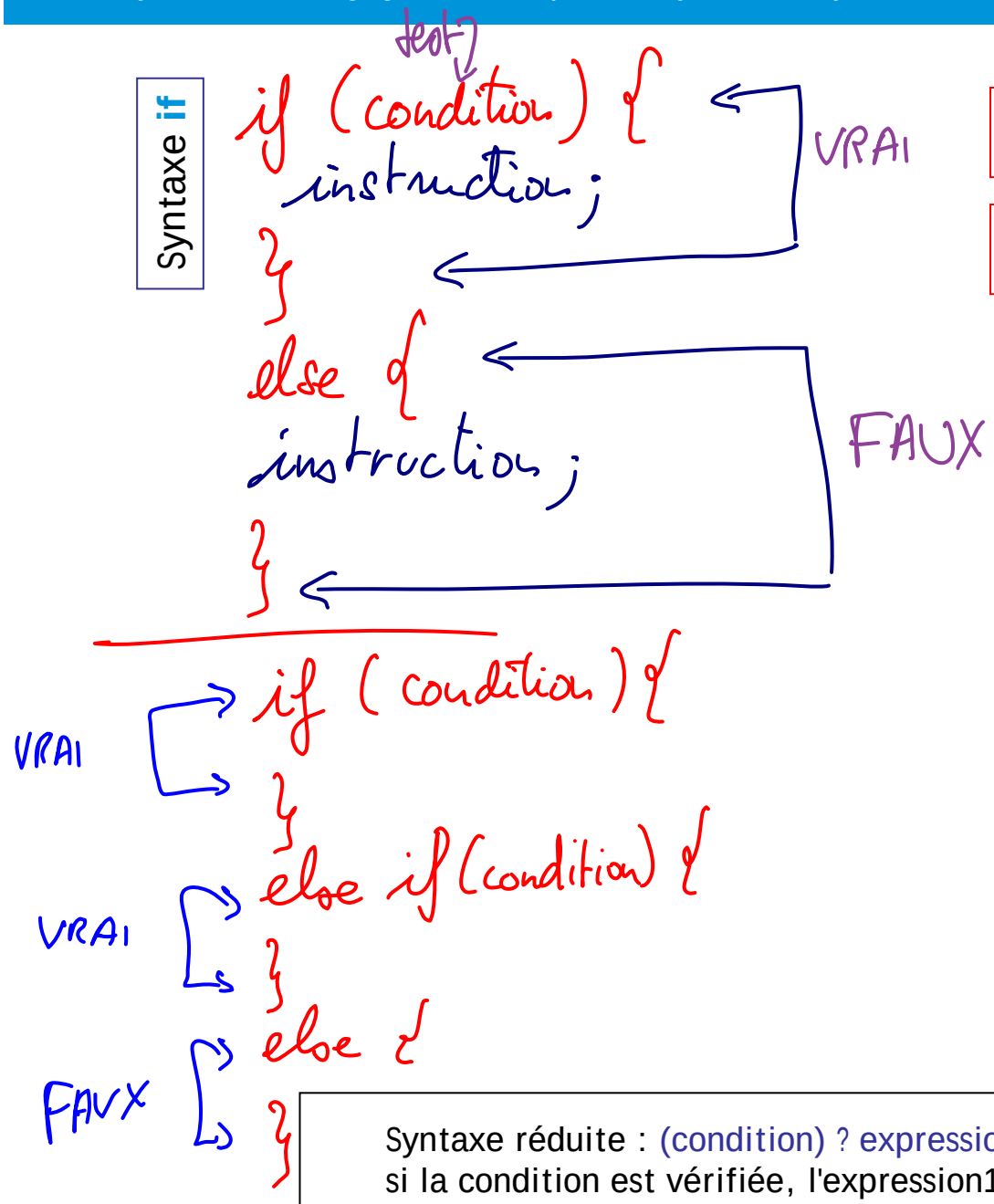
Concatenation

Texte

variable

Test Conditionnel if

Syntaxe **if**



Si le test conditionnel est vérifié, alors ces instructions sont exécutées.

Sinon, ces autres instructions sont exécutées.

Syntaxe réduite : (condition) ? expression1 : expression2
si la condition est vérifiée, l'expression1 est exécutée, sinon l'expression2

On souhaite afficher les nombres pairs entre 1 et *N*.

Schéma d'exécution : Algorithme
Pour cela, déclarez la limite *N*.

L'instruction **for** :
(1) initialise une variable *i* à 0 ;
(2) effectue un test conditionnel, à savoir si *i*<=*N* ;
(3) exécute une série d'instructions séquentiellement;
(4) incrémente la variable *i*.

Dans la boucle for, effectuez un test conditionnel **if**, pour comparer le reste de la division de *i* par 2 (opération modulo) à 0.
Affichez le nombre pair à l'aide la méthode *window.alert( )*.

```
N=10;
for (i=1; i<=N; i++){
  reste = i%2;
  if (reste==0){
    window.alert("Le nombre : "+i+" est pair");
  }
}
```

Pourquoi les fonctions ?

Lorsqu'une suite d'instructions (de calculs) est souvent réalisé, plutôt que de les reprogrammer à chaque fois, on fait appel à une fonction, qui exécute les instruction, et retourne le résultat (ou affiche un résultat). Il est possible d'appeler la fonction en lui passant des paramètres.

Syntaxe **function**

Nom de la fonction

```
function nom_fonction (argument1, argument2, ...) {
  instruction;

  return valeur_retournee;
}
```

Liste d'arguments (valeurs affectées à des variables) utiles pour l'exécution de la fonction. Les valeurs peuvent être des nombres, des chaînes de texte ou des objets.

La valeur de la *variable* est retournée par **return**

valeurs transmise à la fonction

Appel d'une fonction ?

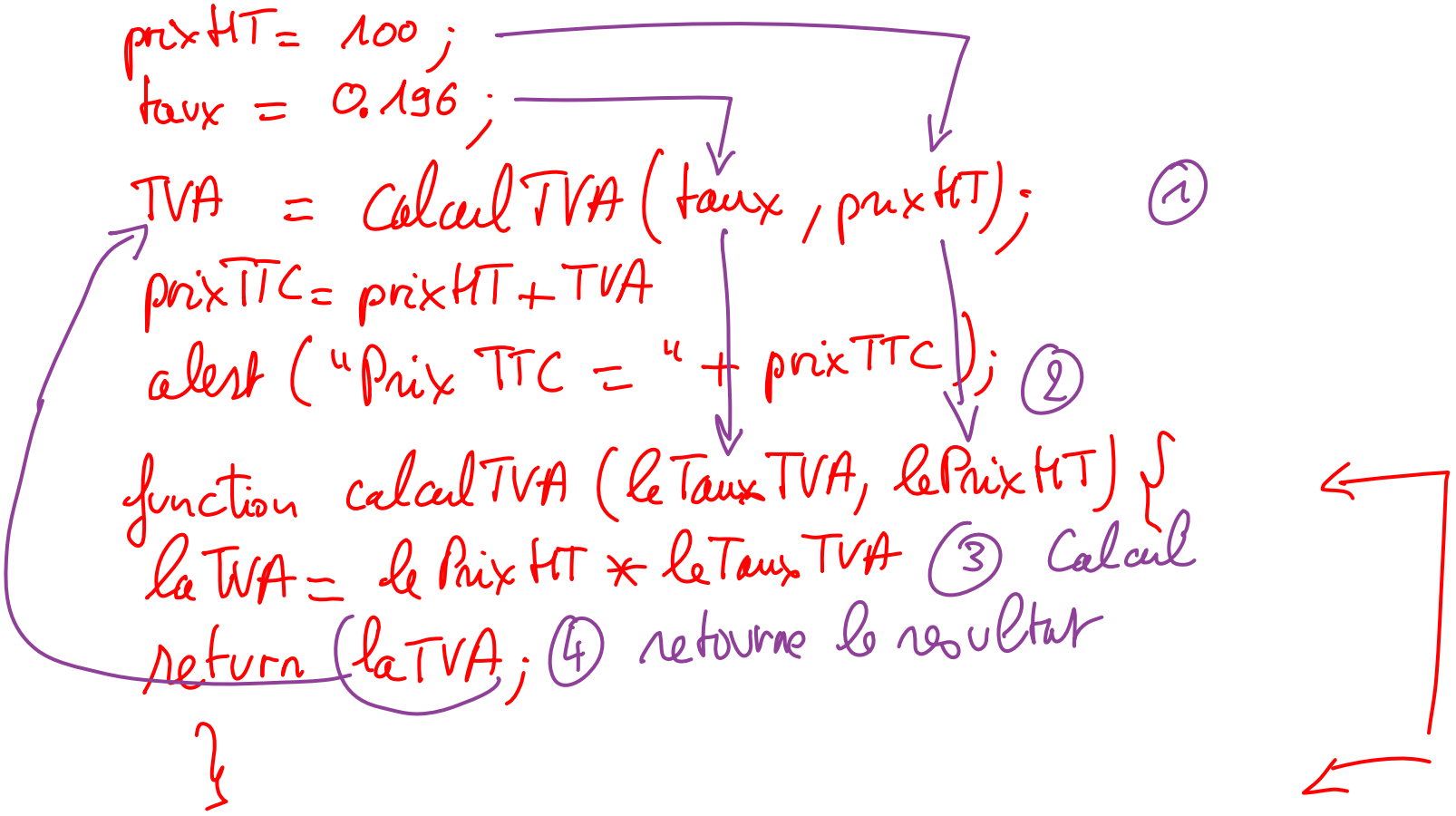
```
resultat = nom_fonction (valeur1, valeur2, ...);
```

↑ variable

On souhaite calculer la part de T.V.A (variable *laTVA*) sur un prix hors taxe (variable *lePrixHT*), puis le prix TTC de l'article (variable*lePricTTC*).

Schéma d'exécution : Algorithme

1. Déclaration des variables, et d'une valeur initiale (le taux de TVA, le prix hors taxe, …)
2. Appel d'une fonction nommée *calculTVA()*, par ex., avec passage des arguments adéquats.
3. Ecriture de la fonction *calculTVA()*, réception des arguments (taux et prix hors taxe), calcul de la TVA, retour de la TVA calculée à l'aide de *return*.
4. Affichage des résultats : prix hors taxe, TVA et prix TTC par la méthode *windows.alert()*.



L'objet **Date** est un objet du noyau JavaScript, c'est à dire qu'il n'y a pas de balises HTML équivalente, cet objet est purement JavaScript.

Un **constructeur de date** permet la déclaration (ou *création*, *instance*) d'un objet de type **Date**:

• Pour affecter dans la variable *laDate* le jour et l'heure (d'après l'horloge du PC) :

```
var laDate = new Date();
```

variable ↔ objet de type Date ; Date() → fonction ≡ Constructeur

• Pour affecter une date autre que celle du moment. Jours et mois sont comptés à partir de 0 et non de 1.

```
laDate = new Date(2006, 1, 4, 17, 0, 0);
```

année, mois, Jour, heure, mn, s

Pour chaque objet de type **Date**, des propriétés sont lues ou écrites, des méthodes sont appliquées. Soit la variable *laDate* :

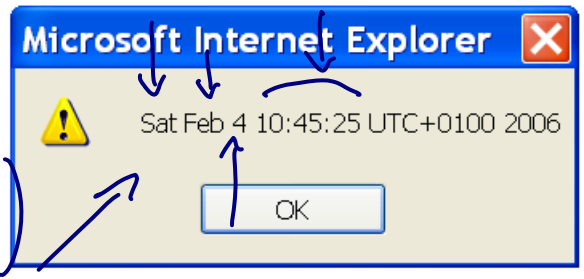
```
objet.propriete
objet.methode()
```

} Syntaxe POO Programmation Orientée Objet.

laDate.propriete → lire / écrire

laDate.methode()

alert(laDate)



Méthodes de l'objet de type Date en lecture : lecture du mois, etc…, le format est *getNomMethode*( )

laDate = new Date( )

lHeure = laDate.getHours( );

Méthode	Description	Type de valeurs retournée
getDate()	Permet de récupérer la valeur du jour du mois	L'objet retourné est un entier (entre 1 et 31) qui correspond au jour du mois:
getDay()	Permet de récupérer la valeur du jour de la semaine pour la date spécifiée	L'objet retourné est un entier qui correspond au jour de la semaine: • 0: dimanche • 1: lundi ...
getFullYear()	Permet de récupérer la valeur de l'anné sur 4 chiffres pour la date passée en paramètre	L'objet retourné est un entier qui correspond à l'année (XXXX) : 2006
getHours()	Permet de récupérer la valeur de l'heure	L'objet retourné est un entier (entre 0 et 23) qui correspond à l'objet Date.
getMilliseconds()	Permet de récupérer le nombre de millisecondes	L'objet retourné est un entier (entre 0 et 999) qui correspond aux millisecondes de l'objet passé en paramètre.
getMinutes()	Permet de récupérer la valeur des minutes	L'objet retourné est un entier (entre 0 et 59) qui correspond aux minutes de l'objet Date.
getMonth()	Permet de récupérer le numéro du mois	L'objet retourné est un entier (entre 0 et 11) qui correspond au mois : • 0: janvier • 1: février ...
getSeconds()	Permet de récupérer le nombre de secondes	L'objet retourné est un entier (entre 0 et 59) qui correspond aux secondes de l'objet passé en paramètre.
getTime()	Permet de récupérer le nombre de millisecondes depuis le 1er janvier 1970	L'objet retourné est un entier. Cette méthode est très utile pour convertir des dates, soustraire ou ajouter deux dates, etc.
getTimezoneOffset()	Retourne la différence entre l'heure locale et l'heure GMT (Greenwich Mean Time)	L'objet retourné est un entier, il représente le nombre de **minutes** de décalage
getYear()	Permet de récupérer la valeur de l'anné sur 2 chiffres pour l'objet Date.	L'objet retourné est un entier qui correspond à l'anné (XX) :

Source : http://www.commentcamarche.net/JavaScript/jsdate.php3

Méthodes de l'objet de type Date en écriture : lecture du mois, etc…, le format est *setNomMethode*( )

Méthode	Description	Type de valeur en paramètre
setDate(X)	Permet de fixer la valeur du jour du mois	Le paramètre est un entier (entre 1 et 31) qui correspond au jour du mois
setDay(X)	Permet de fixer la valeur du jour de la semaine	Le paramètre est un entier qui correspond au jour de la semaine: • 0: dimanche • 1: lundi ...
setHours(X)	Permet de fixer la valeur de l'heure	Le paramètre est un entier (entre 0 et 23) qui correspond à l'heure
setMinutes (X)	Permet de fixer la valeur des minutes	Le paramètre est un entier (entre 0 et 59) qui correspond aux minutes
setMonth(X)	Permet de fixer le numéro du mois	Le paramètre est un entier (entre 0 et 11) qui correspond au mois: • 0: janvier • 1: février ...
setTime(X)	Permet d'assigner la date	Le paramètre est un entier représentant le nombre de millisecondes depuis le 1er janvier 1970

Source : http://www.commentcamarche.net/JavaScript/jsdate.php3

L'horloge digitale : Afficher l'heure sous la forme *hh:mm:ss*

Schéma d'exécution : Algorithme

1. Déclaration d'une variable *laDate*, instance de l'objet **Date()**
2. Lecture de l'heure, des minutes et des secondes à l'aide des méthodes appropriées *getNomMethode*( ). Le résultat de ces lectures est affecté à une variable, soit les variables *lHeure*, *lesMinutes* et *lesSecondes*.
3. Construction d'une chaîne de texte avec le format de l'horloge digitale souhaitée, par concaténation des variables précédentes et du texte ":" .
4. Attention, il conviendra de rajouter un 0 initiale lorsque l'heure, les minutes ou les secondes sont inférieures à 10. En effet, l'horloge doit s'afficher comme 08:05:03 et non 8:5:3, par ex.

Démo

```
laDate = new Date();
lheure = laDate.getHours();
lesMinutes = laDate.getMinutes();
lesSecondes = laDate.getSeconds();
lHorloge = lheure + ":" + lesMinutes + ":" + lesSecondes;
alert(lHorloge);
```

en POO
objet.methode()

L'objet String est un objet du noyau JavaScript, c'est à dire qu'il n'y a pas de balises HTML équivalente, cet objet est purement JavaScript. C'est une **chaîne de texte**.
⇨ Une variable peut contenir un nombre, ou du **texte**, dans ce cas la variable est du type **String**.

Une variable de type String est déclarée de la manière suivante:

```
leNom = "Chuchoffel";
```

variable de type String

⇨ Cette variable est implicitement un objet **String**, en raison du contenu de type texte affecté à cette variable. **Rappel**: *les variables ne sont pas typées!*

Pour chaque objet **String**, et donc chaîne de texte, des **propriétés** sont lues ou écrites, des **méthodes** sont appliquées:

```
objet.propriete   ⇄ lire / écriture
objet.methode()
```

leNom.propriete

Propriétés de l'objet String :
Il n'existe qu'une seule propriété, la propriété **length**, qui retourne le nombre de caractère de la chaîne de texte.

```
var nbCar = leNom.length;
```

Méthodes de l'objet String :

Des méthodes permettent de **transformer** une chaîne de texte : en majuscule, en minuscule, d'extraire une sous-chaîne de texte...

Méthode	description
Chaine.anchor ("nom_a_donner");	Transforme le texte *Chaine* en ancrage HTML.
Chaine.big()	Augmente la taille de la police.
Chaine.blink()	Transforme la chaîne en texte clignotant.
Chaine.bold()	Met le texte en **gras** (balise <B>).
Chaine.charAt(position)	Retourne le caractère situé à la position donnée en paramètre
Chaine.charCodeAt(position)	Renvoie le code *Unicode* du caractère situé à la position donnée en paramètre
concat(chaîne1, chaîne2[, ...])	Permet de concaténer les chaînes passées en paramètre, c'est-à-dire de les joindre bout à bout.
Chaine.fixed()	Transforme la Chaine en caractères de police fixe (balise <TT>)
Chaine.fontcolor(couleur)	Modifie la couleur du texte (admet comme argument la couleur en hexadécimal ou en valeur littérale)
Chaine.fontsize(Size)	Modifie la taille de la police, en afectant la valeur passée en paramètre
Chaine.fromCharCode(code1 [, code2, ..])	Renvoie une chaîne de caractères composée de caractères correspondant au (x) code(s) *Unicode* donné(s) en paramètre.
Chaine.indexOf(sous-chaîne, position)	Retourne la position d'une sous-chaîne (lettre ou groupe de lettres) dans une chaîne de caractère, en effectuant la recherche de gauche à droite, à partir de la position spécifiée en paramètre.
Chaine.italics()	Transforme le texte en *italique* (balise <I>)
Chaine.lastIndexOf(sous-chaîne, position)	La méthode est similaire à *indexOf()*, à la différence que la recherche se fait de droite à gauche: Retourne la position d'une sous-chaîne (lettre ou groupe de lettres) dans une chaîne de caractère, en effectuant la recherche de droite à gauche, à partir de la position spécifiée en paramètre.
Chaine.link(URL)	Transforme le texte en hypertexte (balise <A href>)
Chaine.small()	Diminue la taille de la police
Chaine.strike()	Transforme le texte en ~~texte barré~~ (balise <strike>)
Chaine.sub()	Transforme le texte en $_{\text{indice}}$ (balise <sub>)
Chaine.substr(position1, longueur)	La méthode retourne une sous-chaîne commençant à l'index dont la position est donnée en argument et de la longueur donnée en paramètre.
Chaine.substring(position1, position2)	La méthode retourne la sous-chaîne (lettre ou groupe de lettres) comprise entre les positions 1 et 2 données en paramètre.
Chaine.sup()	Transforme le texte en $^{\text{exposant}}$ (balise <sup>).
Chaine.toLowerCase()	Convertit tous les caractères d'une chaîne en minuscule.
Chaine.toSource()	Renvoie le code source de création de l'objet.
Chaine.toUpperCase()	Convertit tous les caractères d'une chaîne en majuscule.
Chaine.valueOf()	Renvoie la valeur de l'objet String.

var leNom = "Christoffel";

leNomM = leNom.toUpperCase();

Méthode split("?")

adrCourriel = "nom@domaine";

nom domaine

Split permet de découper à partir d'un caractère

resultat = adrCourriel.split("@");

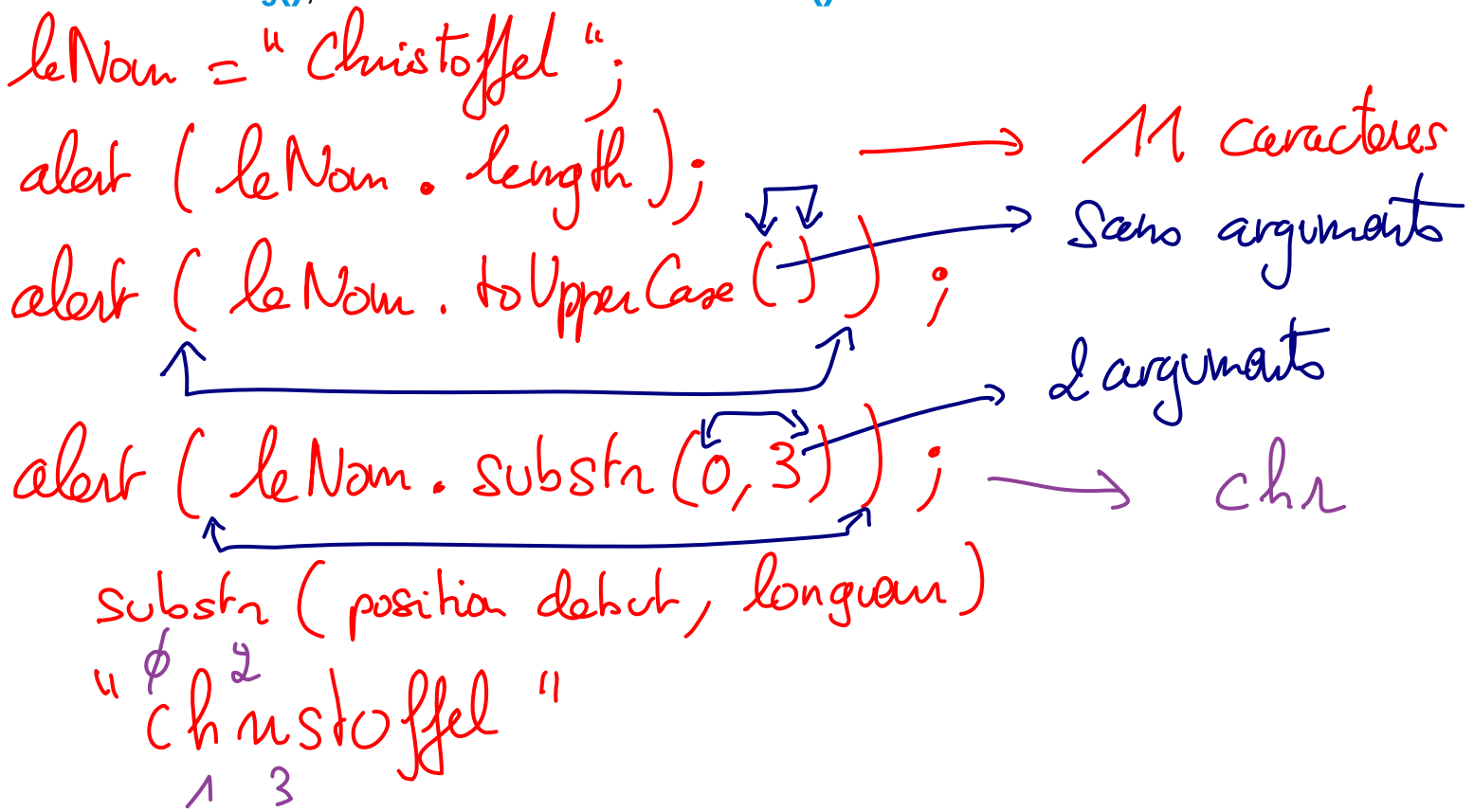
↳ tableau JS : objet Array

Source : http://www.commentcamarche.net/JavaScript/jsstring.php3

Transformer une chaîne de texte en majuscule, puis en extraire les 3 premières lettres, les 3 dernière.

Schéma d'exécution :

1. Déclaration d'une variable *leNom*, et affectation d'une valeur. Implicitement, la variable sera de type **String**.
2. Transformer la chaîne de texte en majuscule, méthode **toUpperCase()**, et affichage du résultat par la méthode **windows.alert()**.
3. Extraire les 3 premières lettres à l'aide la méthode **substr()**, puis les 3 dernières à l'aide de la méthode **substring()**, variante de la méthode **substr()**. Afficher les résultats.



L'objet **Array** est un objet du noyau JavaScript , c'est à dire qu'il n'y a pas de balises HTML équivalente, cet objet est purement JavaScript. C'est un **tableau de données**.
Un **constructeur de tableau** permet la déclaration (ou *création*, *instance*) d'un objet de type **Array :**

```
var leTab = new Array();
```

Il n'est pas nécessaire de préciser le nombre d'éléments du tableau, sa **dimension**. Celle-ci sera automatiquement ajustée par JavaScript en fonction du contenu du tableau. Affectation de valeurs aux éléments du tableau :

```
leTab[0] = "Chistoffel";
leTab[1] = new Date();
```

POO

Mais, il est aussi possible d'affecter directement un contenu à chaque cellule du tableau lors de la déclaration:

```
leTab = new Array("Chistoffel", "03.90.24.06.35", 3,..)
```

Noter que pour accéder à un élément du tableau, on utilise les **[ ]**:

```
leTab[i]
```

Pour chaque tableau, des **propriétés** sont lues ou écrites, des **méthodes** sont appliquées:

```
leTab.propriete , leTab.methode()
```

Propriétés de l'objet Array :
Il n'existe qu'une seule propriété, la propriété **length**, qui retourne le nombre d'éléments du tableau, à savoir sa **dimension** :

```
alert(leTab.length);
```

Méthodes de l'objet Array :

Des méthodes permettent de **transformer** le tableau : trier les éléments du tableau en prdre croissant, de joindre les éléments du tableau en une seule chaîne de texte, ...

Méthode	description
concat(tab1, tab2[, tab3, ...])	Cette méthode permet de concaténer plusieurs tableaux, c'est-à-dire de créer un tableau à partir des différents tableaux passés en paramètre.
join(tableau) ou **Tableau.join()**	Cette méthode renvoie une chaîne de caractères contenant tous les éléments du tableau.
pop(tableau) ou **Tableau.pop()**	Cette méthode supprime le dernier élément du tableau et retourne sa valeur.
Tableau.push(valeur1[, valeur2, ...])	Cette méthode ajoute un ou plusieurs éléments au tableau.
Tableau.reverse()	Cette méthode inverse l'ordre des éléments du tableau.
Tableau.shift()	Cette méthode supprime le premier élément du tableau.
Tableau.slice()	Cette méthode renvoie un tableau contenant une partie (extraction) des éléments d'un tableau.
Tableau.splice()	Cette méthode ajoute/retire des éléments d'un tableau.
Tableau.sort()	Cette méthode permet de trier les éléments d'un tableau.
Tableau.unshift(valeur1[, valeur2, ...])	Cette méthode renvoie le code source qui a permis de créer l'objet *Array*.
Tableau.toString()	Cette méthode renvoie la chaîne de caractères correspond à l'instruction qui a permis de créer l'objet *Array*.
Tableau.unshift()	Cette méthode permet dajouter un ou plusieurs élément au début du tableau.
Tableau.valueOf	Cette méthode retourne tout simplement la valeur de l'objet *Array* auquel elle fait référence.

leTab = new Array ("Christoffel", "Eric");

lIdentite = leTab.join(" "); espace

variable ⇓

objet de type String ⟹ "Christoffel Eric"

Source : http://www.commentcamarche.net/JavaScript/jsarray.php3

Une chaîne de texte contient une adresse courriel, du type *prenom_nom@domaine.com*. Décomposer cette adresse courriel et afficher séparément le destinataire et le nom de domaine.

Schéma d'exécution :

1. Déclaration d'une variable de type **String** nommée *adresseCourriel*, et affectation d'une valeur.
2. Décomposition de cette variable par la méthode **split(".")** de l'objet **String**. Le résultat de cette décomposition est affecté dans une variable nommée *resultat*, de type **Array**.
3. Afficher la dimension du tableau *resultat*.
4. Afficher le résultat de la décomposition: destinataire et nom de domaine.

```
adresseCourriel = "christof@physique.unistra.fr";
resultat        = adresseCourriel.split("@");
alert( resultat.length );
alert( resultat.join("\n") );
```

objet Array ⇐ resultat

"\n" : saut de ligne

Il est possible **de créer des objets**, qui n'existent pas sous Javascript (ni dans le noyau javascript, ni dans les objets liés aux pages web). Ces objets nous seront utiles pour créer des **données sous forme d'une base de données**.

→ écriture dynamique d'une page HTML

Nommons arbitrairement ce nouvel objet **Objet**, qui devient un objet au même titre que les objets **Date**, **String** et **Array**, vus précédemment.
A cet objet **Objet** nous lui affectons des **propriétés**, nommées *propriete1*, *propriete2*, etc... Tout comme l'objet **String** possède une propriété *length*.

```
var laDate = new Date();
leMois = laDate.getMouth();
```

```
Objet.propriete
Objet.methode()
```

Déclaration de l'objet :
Effectuons une **instance** de cet objet, comme nous le ferions pour un nouveau tableau **Array**, dans une variable :

```
monObjet = new Objet(lArgument1, lArgument2, -- )
```

Ici, nous passons une liste **d'arguments** à l'objet **Objet** lors de la déclaration.

Le constructeur d'objet :
C'est une **fonction**, qui **reçoit les arguments** passés lors de la déclaration, et qui **les affecte aux propriétés** de ce nouvel objet:

```
function Objet(laValeurRecue1, laValeurRecue2, --- ){
this.propriete1 = laValeurRecue1;
this.propriete2 = laValeurRecue2;
}
```

création de la propriete1

```
alert(monObjet.propriete2);
```

Ici, **this** fait référence à l'objet en cours de déclaration, et permet d'accéder à une propriété donnée, et d'y affecter une valeur passée en argument.

Remarque: il existe également une fonction Array, tel que function Array(){...} , appartenant au noyau javascript, et donc non visible pour nous, qui est appelée lors d'une instance d'un objet Array.

Nous souhaitons créer une base de données d'employés d'une entreprise sous javascript. Pour cela, nous créerons un objet personnalisé nommé **employe**, de propriétés le *nom*, le *prénom* et le *téléphone*. Mais plusieurs employés sont dans l'entreprise! Nous créerons alors une variable **lesEmployes** de type **Array** (tableau javascript), dont chaque élément de ce tableau javascript sera une instance de l'objet personnalisé **employe**.

Schéma d'exécution :

1. Déclaration de la variable *lesEmployes*, tableau javascript, comme une instance de l'objet Array
2. Chaque employé est un élément du tableau lesEmployés. Le premier employé que nous déclarons est donc l'élément de rang 0 du tableau *lesEmployes*, soit l'élément *lesEmployés[0]*.
3. Cet élément est une instance de l'objet personnalisé **employe**. Lors de l'instance, les données de l'employé (nom, prénom et téléphone) sont passées en argument d'appel de la fonction **employe** (du constructeur).
4. Ecriture de la fonction **employe**, constructeur de l'objet personnalisé **employe**.
5. Affichage du prénom de l'employé de rang 0 (par ex.)

```
1) lesEmployes = new Array( );  ——→ Array JS
2)&3) lesEmployes[0] = new employe ("Christoffel", "Eric", "03 90.24 06.35"),
4) function employe (leNom, lePrenom, leTel) {
     this.nom = leNom;
     this.prenom = lePrenom;
     this.tel = leTel;
   }
5) alert (lesEmployes[0].prenom) ——→ Eric
```

Propriétés et Méthodes :
Le nouvel objet créé ne semble posséder que des propriétés. Un objet sans méthodes aurait des applications limitées!

Pour tous les objets du noyau javascript, ainsi que pour les objets personnalisés, il existe les propriétés **constructor** et **prototype** qui permettent respectivement :
- de retrouver le constructeur d'un objet,
- mais surtout de lui rajouter de nouvelles propriétés et méthodes.

Les instructions suivantes permettent d'afficher le constructeur, puis de créer une nouvelle propriété, et enfin une nouvelle méthode :

1) Constructor

```
alert(lesEmployes.constructor);
alert(lesEmployes[0].constructor);
```

affiche la fonction :
function Array(){...}
function employe(){...}

2) prototype

```
employe.prototype.nomPropriete = valeurDefaut;
employe.prototype.statut = "Enseignant";
employe.prototype.nomMethode = nomFonction;

function nomFonction(){
    instruction;
}

lesEmployes[0].nomMethode();
```