

Conception de sites web dynamique

***HTML – CSS – JAVASCRIPT – PHP
BASE DE DONNÉES***

IF04U050

David Bouchet

david.bouchet.paris5@gmail.com

JAVASCRIPT ***1^{re} partie***

A yellow square containing the letters 'JS' in a bold, dark blue font, representing the JavaScript logo.

JS

Qu'est-ce que JavaScript ?

Langage de programmation

Langage de script = Langage interprété

L'interpréteur se trouve dans le navigateur

Objectif = dynamiser html :

- Adapter la mise en page
- Ouvrir de nouvelles fenêtres
- Réagir à des actions de l'utilisateur
- Traiter des infos saisies par l'utilisateur

localement,
sur le client

Inclusion de codes JavaScript (1)

Directement dans le code HTML à l'aide de l'élément *script* :

```
<head>
  <script>
    // Insérer du code JavaScript ici.
  </script>
  ...
</head>
<body>
  ...
  <script>
    // Insérer du code JavaScript ici.
  </script>
  ...
</body>
```

Les balises
<script>...</script>
peuvent se placer
n'importe où dans
le fichier HTML.

Inclusion de codes JavaScript (2)

Dans des fichiers externes à l'aide de l'élément *script* et de l'attribut *src*.

```
<head>
  <script src="fichier1.js"></script>
</head>
<body>
  ...
  <script src="fichier2.js"></script>
  ...
</body>
```

Les balises
<script>...</script>
peuvent se placer
n'importe où dans
le fichier HTML.

Les fichiers "fichier1.js" et "fichier2.js" contiendront uniquement du code JavaScript.

Il est plus prudent de placer l'élément *script* à la fin du fichier HTML. Ceci permettra au code JavaScript de s'exécuter une fois que la page aura été complètement chargée par le navigateur.

Commentaires

Il existe deux types de commentaires :

- `//` texte de commentaire sur une ligne
- `/*` texte de commentaire `*/`

```
// Ceci est un commentaire sur une seule ligne.
```

```
/* Ceci est un commentaire sur plusieurs lignes.  
   Il peut être aussi long que souhaité,  
   mais il ne peut pas contenir un autre  
   commentaire du même type. */
```

Types primitifs

6 types primitifs

- `number` (nombres entiers ou flottants)
- `boolean` (booléens)
- `string` (chaîne de caractères)
- `undefined` (variable indéfinie)
- `null` (variable définie, mais sans valeur)
- `symbol` (nouveau type, nous ne l'utiliserons pas dans ce cours)

Déclaration d'une variable (1)

Déclaration à l'aide du mot clé **var** quel que soit le type.

```
var a;           // Type undefined
var b = 250;     // Type number
var c = "Chaîne"; // Type string
```

Le mot clé **typeof** renvoie le type d'une variable.

```
typeof a; // Renvoie la chaîne "undefined";
typeof b; // Renvoie la chaîne "number";
typeof c; // Renvoie la chaîne "string"
```

Déclaration à l'aide du mot clé **let**.

Similaire à **var** mais avec une portée réduite au bloc.

```
let a;           // Type undefined
let b = 250;     // Type number
let c = "Chaîne"; // Type string
```


Déclaration d'une variable (2)

```
for (let i = 0; i < 3; i++)  
  console.log("i =", i);  
  
// console.log("i après la boucle = ", i);  
console.log("typeof i =", typeof i);  
  
for (var j = 0; j < 3; j++)  
  console.log("j = ", j);  
  
console.log("j après la boucle =", j);  
console.log("typeof j =", typeof j);
```

```
i = 0  
i = 1  
i = 2  
typeof i = undefined  
j = 0  
j = 1  
j = 2  
j après la boucle = 3  
typeof j = number
```

Type *number*

Type utilisé pour les données numériques :

- Gestion de nombres entiers et de nombres flottants.
- Possède trois symboles :
 - **+Infinity** (infinité positive).
 - **-Infinity** (infinité négative).
 - **NaN** (Not A Number – Entité qui n'est pas un nombre.)

```
var a = 250;           // Nombre entier
var b = 250.25;        // Nombre flottant
var c = -19824.125;    // Nombre flottant
var d = -Infinity;     // Moins l'infini
var e = NaN;           // Not A Number
var f = 0xff;          // Notation hexadécimale (nombre entier)
var h = 0b1001101;     // Notation binaire (nombre entier)
```

Type *string*

Type utilisé pour les données textuelles :

- Concaténation à l'aide de l'opérateur `+`.
- Longueur de la chaîne à l'aide de la propriété *length*.
- Récupération d'un caractère à l'aide de *[n]* ou *charAt()*.
- Caractère d'échappement : \

```
var a = "Bonjour";           // a = "Bonjour"
var b = "David";             // b = "David"
var c = a + " " + b + "\n";  // c = "Bonjour David\n"
var d = a[0];                // d = "B"
var e = a[1];                // e = "o"
var f = a.charAt(2);         // f = "n"
var g = a.length;           // g = 7
var h = b.length;           // h = 5
var i = c.length;           // i = 14
```

Type *string* – Caractères spéciaux

Caractère	Signification
\0	Octet null
\b	Retour arrière
\f	Saut de page
\n	Nouvelle ligne
\r	Retour chariot
\t	Tabulation
\v	Tabulation verticale
\'	Apostrophe ou guillemet droit simple
\"	Guillemet droit double
\\	Barre oblique inversée

Type *boolean*

Type utilisé pour les booléens :

- Seulement deux valeurs possibles : *true* et *false*

```
var a = true;  
var b = false;
```

Type *null*

Type utilisé pour des variables définies dont on veut explicitement exprimer qu'elles ne possèdent pas de valeur particulière.

- Seulement une valeur possible : *null*
- Ce type est souvent utilisé en valeur de retour lorsqu'un objet est attendu mais qu'aucun objet ne convient.

```
var a = null;
```

Type *undefined*

Type utilisé pour des variables qui ne possèdent pas de valeur.

```
var a;  
  
typeof a;    // Renvoie la chaîne "undefined";  
typeof z;    // Renvoie la chaîne "undefined";
```

Afficher des messages

Il existe plusieurs façons d'afficher des messages. Nous verrons les deux plus couramment utilisées.

Afficher un message dans la fenêtre console du navigateur :

```
console.log(message);
```

Afficher un message dans une boîte de dialogue d'alerte :

```
alert(message);
```

Exemple :

```
console.log("Hello"); // Affiche "Hello" dans la console
var a = 200;
console.log(a);        // Affiche "200" dans la console
alert("Hi !");         // Affiche "Hi !" dans une fenêtre d'alerte.
alert(a);              // Affiche "200" dans une fenêtre d'alerte.
```


Typage dynamique

Une variable peut changer de type à tout moment.

```
var a = "Bonjour";  
console.log(typeof a);      // Affiche "string"  
  
a = 5;  
console.log(typeof a);      // Affiche "number"  
  
a = true;  
console.log(typeof a);      // Affiche "boolean"  
  
a = undefined;  
console.log(typeof a);      // Affiche "undefined"
```

Conversion implicite (*string* ↔ *number*)

Conversion implicite d'un nombre en chaîne

```
x = "La réponse est " + 42;    // x = "La réponse est 42"  
y = 42 + " est la réponse";    // y = "42 est la réponse"  
z = "20" + 7;                 // z = "207"
```

Conversion implicite d'une chaîne en nombre

```
x = "20" - 7;    // x = 13  
y = "20" * 2;    // y = 40  
z = "20" / 2;    // z = 10
```

Opérateurs d'affectation

Nom	Opérateur composé	Signification
Affectation	$x = y$	$x = y$
Affectation après addition	$x += y$	$x = x + y$
Affectation après soustraction	$x -= y$	$x = x - y$
Affectation après multiplication	$x *= y$	$x = x * y$
Affectation après division	$x /= y$	$x = x / y$
Affectation du reste	$x \% = y$	$x = x \% y$
Affectation après exponentiation	$x ** = y$	$x = x ** y$
Affectation après décalage à gauche	$x << = y$	$x = x << y$
Affectation après décalage à droite	$x >> = y$	$x = x >> y$
Affectation après décalage à droite non signé	$x >>> = y$	$x = x >>> y$
Affectation après ET binaire	$x \& = y$	$x = x \& y$
Affectation après OU exclusif binaire	$x \wedge = y$	$x = x \wedge y$
Affectation après OU binaire	$x \mid = y$	$x = x \mid y$

Opérateurs de comparaison

Opérateur	Description	Exemples qui renvoient true
Égalité (==)	Renvoie true si les opérandes sont égaux après conversion en valeurs de mêmes types.	<code>3 == var1</code> <code>"3" == var1</code> <code>3 == '3'</code>
Inégalité (!=)	Renvoie true si les opérandes sont différents.	<code>var1 != 4</code> <code>var2 != "3"</code>
Égalité stricte (===)	Renvoie true si les opérandes sont égaux et de même type. Voir Object.is() et égalité de type en JavaScript .	<code>3 === var1</code>
Inégalité stricte (!==)	Renvoie true si les opérandes ne sont pas égaux ou s'ils ne sont pas de même type.	<code>var1 !== "3"</code> <code>3 !== '3'</code>
Supériorité stricte (>)	Renvoie true si l'opérande gauche est supérieur (strictement) à l'opérande droit.	<code>var2 > var1</code> <code>"12" > 2</code>
Supériorité ou égalité (>=)	Renvoie true si l'opérande gauche est supérieur ou égal à l'opérande droit.	<code>var2 >= var1</code> <code>var1 >= 3</code>
Infériorité stricte (<)	Renvoie true si l'opérande gauche est inférieur (strictement) à l'opérande droit.	<code>var1 < var2</code> <code>"2" < "12"</code>
Infériorité ou égalité (<=)	Renvoie true si l'opérande gauche est inférieur ou égal à l'opérande droit.	<code>var1 <= var2</code> <code>var2 <= 5</code>

```
var var1 = 3  
var var2 = 4
```

Opérateurs binaires (1)

Opérateur	Utilisation	Description
AND (ET) binaire	$a \& b$	Renvoie un 1 à chaque position binaire pour laquelle les bits des deux opérandes sont à 1.
OR (OU) binaire	$a \mid b$	Renvoie un zéro à chaque position binaire pour laquelle au moins un des bits des deux opérandes est à 0.
XOR (OU exclusif) binaire	$a \wedge b$	Renvoie un zéro à chaque position binaire pour laquelle les bits sont les mêmes (et un 1 pour chacun des bits qui est différent).
NOT (NON) binaire	$\sim a$	Inverse les bits de l'opérande.

Opérateurs binaires (2)

Opérateur	Description	Exemple
Décalage à gauche (<<)	Cet opérateur décale le premier opérande d'un nombre de bits donné sur la gauche. Les bits en trop sont ignorés et des bits à zéro sont introduits à droite.	9<<2 renvoie 36, car 1001, décalé de 2 bits à gauche, devient 100100, dont la représentation en base 10 est 36.
<u>Décalage à droite avec propagation du signe (>>)</u>	Cet opérateur décale le premier opérande d'un nombre de bits donné sur la droite. Les bits en trop sont ignorés et des bits correspondants au bit de signe sont introduits à gauche.	9>>2 renvoie 2, car 1001, décalé de 2 bits à droite, devient 10 représentant 2. De même -9>>2 renvoie -3, car le signe est préservé.
Décalage à droite avec zéros (>>>)	Cet opérateur décale le premier opérande d'un nombre de bits donné sur la droite. Les bits en trop sont ignorés et des bits à 0 sont introduits à gauche.	19>>>2 renvoie 4, car 10011, décalé de 2 bits, devient 100 qui représente 4. Pour les nombres positifs, cet opérateur et l'opérateur précédent renvoient les mêmes résultats.

Opérateurs logiques (1)

Opérateur	Usage	Description
ET logique (<code>&&</code>)	<code>expr1 && expr2</code>	Renvoie <code>expr1</code> s'il peut être converti à <code>false</code> , sinon renvoie <code>expr2</code> . Dans le cas où on utilise des opérandes booléens, <code>&&</code> renvoie <code>true</code> si les deux opérandes valent <code>true</code> , <code>false</code> sinon.
OU logique (<code> </code>)	<code>expr1 expr2</code>	Renvoie <code>expr1</code> s'il peut être converti à <code>true</code> , sinon renvoie <code>expr2</code> . Dans le cas où on utilise des opérandes booléens, <code> </code> renvoie <code>true</code> si l'un des opérandes vaut <code>true</code> , si les deux valent <code>false</code> , il renvoie <code>false</code> .
NON logique (<code>!</code>)	<code>!expr</code>	Renvoie <code>false</code> si son unique opérande peut être converti en <code>true</code> , sinon il renvoie <code>true</code> .

Les exemples d'expressions qui peuvent être converties à **false** sont celles qui sont évaluées à **null**, **0**, **NaN**, la chaîne de caractères vide (`""`), ou **undefined**.

Opérateurs logiques (2)

```
var a1 = true && true;    // t && t renvoie true
var a2 = true && false;   // t && f renvoie false
var a3 = false && true;   // f && t renvoie false
var a4 = false && (3 == 4); // f && f renvoie false
var a5 = "Chat" && "Chien"; // t && t renvoie Chien
var a6 = false && "Chat"; // f && t renvoie false
var a7 = "Chat" && false; // t && f renvoie false

var o1 = true || true;   // t || t renvoie true
var o2 = false || true;  // f || t renvoie true
var o3 = true || false;  // t || f renvoie true
var o4 = false || (3 == 4); // f || f renvoie false
var o5 = "Chat" || "Chien"; // t || t renvoie Chat
var o6 = false || "Chat"; // f || t renvoie Chat
var o7 = "Chat" || false; // t || f renvoie Chat

var n1 = !true; // !t renvoie false
var n2 = !false; // !f renvoie true
var n3 = !"Chat"; // !t renvoie false
```


Opérateurs arithmétiques

Opérateurs standards : **+**, **-**, *****, **/**

Autres :

Opérateur	Description	Exemple
Reste (%)	Opérateur binaire. Renvoie le reste entier de la division entre les deux opérandes.	12 % 5 renvoie 2.
Incrément (++)	Opérateur unaire. Ajoute un à son opérande. S'il est utilisé en préfixe (++x), il renvoie la valeur de l'opérande après avoir ajouté un, s'il est utilisé comme opérateur de suffixe (x++), il renvoie la valeur de l'opérande avant d'ajouter un.	Si x vaut 3, ++x incrémente x à 4 et renvoie 4, x++ renvoie 3 et seulement ensuite ajoute un à x.
Décrément (--)	Opérateur unaire. Il soustrait un à son opérande. Il fonctionne de manière analogue à l'opérateur d'incrément.	Si x vaut 3, --x décrémente x à 2 puis renvoie 2, x-- renvoie 3 puis décrémente la valeur de x.
Négation unaire (-)	Opérateur unaire. Renvoie la valeur opposée de l'opérande.	Si x vaut 3, alors -x renvoie -3.
Plus unaire (+)	Opérateur unaire. Si l'opérande n'est pas un nombre, il tente de le convertir en une valeur numérique.	+ "3" renvoie 3. + true renvoie 1.

Opérateur conditionnel ternaire

Syntaxe :

```
condition ? val1 : val2
```

Si la condition vaut *true*, l'opérateur vaudra *val1*. Sinon il vaudra *val2*.

Exemple :

```
var statut = (age >= 18) ? "adulte" : "mineur";
```

Instructions de contrôle (1)

```
if (condition)
{
    instruction1
    ...
    instructionp
}

else
{
    instruction1
    ...
    instructionp
}
```

```
switch (expression)
{
    case val_1: instruction_1; break;
    .....
    case val_n: instruction_n; break;
    default: instructiondefault;
}
```

Exemple :

```
<script>
    var chiffre = 2
    switch (chiffre)
    {
        case 1: alert('Un'); break;
        case 2: alert('Deux'); break;
        case 3: alert('Trois'); break;
        default: alert('Ce n\' est pas un chiffre < 4');
    }
</script>
```

Instructions de contrôle (2)

```
while (condition)
    { instruction_1 ; ..... ; instruction_p }
```

```
do { instruction_1 ; ..... ; instruction_p ;}
    while (condition)
```

```
for (initialisation ; test ; incrémentation)
{   instruction_1 ; ..... ; instruction_p ; }
```

Exemple :

```
<script language="javascript" >
    var i ;
    for (i = 5; i <= 10 ; i++) { alert(i + "\n"); }
</script>
```

Les fonctions (1)

Déclaration :

```
function NomFonction(arg_1, ..., arg_n)
{
    instruction_1;
    .....
    instruction_p;
    return valeur_a_renvoyer;
}
```

Exemple :

```
<script>
    function salutation(prenom)
    {
        alert("Bonjour " + prenom);
    }
</script>
```

Les fonctions (2)

Portée des variables. Ex :

```
<script>
  var x="dehors";
  function essai() {var x="dedans"; alert(x);}
  alert(x); essai();
</script>
```

Tableau des arguments : arguments[]

```
function myConcat(separateur)
{
  var resultat=""; // Initialisation de la boucle.
  for (i = 1; i < arguments.length; i++)
  { resultat += arguments[i] + separateur;}
  return resultat;
}
```

Appel : myConcat(" - ","red","orange","blue")

Retour : "red - orange - blue - "

Objet (1)

Un objet est un ensemble de propriétés et de méthodes.

Une propriété est une variable appartenant à l'objet.

Une méthode est une fonction appartenant à l'objet.

Objets (2)

Différence javascript / langages objets classiques (java, C++...) :

Programmation objet classique :

1) Définition d'une **classe** = Définitions d'un ensemble de propriétés et de méthodes

Ex : classe Rectangle → longueur, largeur, surface()

2) Déclaration d'**objets** = **instances de la classe** :

On donne des valeurs aux propriétés.

Ex : un_rectangle → longueur=5, largeur=3

Programmation objet javascript :

- **Objets définis sans avoir défini de classe au préalable**

Ex : un_rectangle = longueur 5, largeur 3 et surface()

- **Typage dynamique** : on peut ajouter des propriétés plus tard.

Création d'un objet (1)

Création d'un objet à l'aide de *new Object* :

```
var personne_1 = new Object;           // Déclaration de l'objet

personne_1.prenom = "René";           // Définition d'une propriété
personne_1.nom = "Descartes";          // Définition d'une propriété
personne_1.afficher = afficher;        // Définition d'une méthode

personne_1.afficher();                 // Appel d'une méthode

function afficher()                   // Déclaration de la méthode
{
    var x;

    x = "Nom : " + this.nom + "\n";
    x += "Prénom : " + this.prenom;

    alert(x);
}
```

Création d'un objet (2)

Création d'un objet à l'aide d'un initialiseur :

```
var personne_1 =           // Déclaration à l'aide d'un initialiseur
{
    prenom: "René",        // Définition d'une propriété
    nom: "Descartes",      // Définition d'une propriété
    afficher: afficher     // Définition d'une méthode
};

personne_1.afficher();     // Appel d'une méthode

function afficher()        // Déclaration de la méthode
{
    var x;

    x = "Nom : " + this.nom + "\n";
    x += "Prénom : " + this.prenom;

    alert(x);
}
```

Création d'un objet (3)

Création d'un objet à l'aide d'un constructeur :

```
var personne_1 = new Personne("René", "Descartes"); // Appel d'un constructeur

personne_1.afficher(); // Appel d'une méthode

function Personne(nom, prenom) // Déclaration du constructeur
{
    this.nom = nom; // Définition d'une propriété
    this.prenom = prenom; // Définition d'une propriété
    this.afficher = afficher; // Définition d'une méthode
}


function afficher() // Déclaration de la méthode
{
    var x;

    x = "Nom : " + this.nom + "\n";
    x += "Prénom : " + this.prenom;

    alert(x);
}
```

Événements (1)

Un événement permet de lancer une fonction lorsqu'une action particulière s'est produite.

Événement = 

- erreur de l'interpréteur Javascript
- écoulement d'un laps de temps (minuterie)
- action de l'utilisateur



Détection par le navigateur



Traitement par un gestionnaire d'événement :

```
<nomBalise onEvenement="code JS à exécuter">
```

Événements (2)

Événement	Description
onAbort	Chargement d'une image interrompu
onBlur	Perte du focus
onChange	Contenu d'un champ de données modifié
onClick	Clic sur un élément
onDblclick	Double clic sur un élément
onDragdrop	Gilsser-déposer dans la fenêtre du navigateur
onError	Une erreur apparaît pendant le chargement de la page
onFocus	Récupération du focus
onKeyDown	Touche appuyée
onKeyPress	Touche maintenue appuyée

Événements (3)

Événement	Description
onKeyUp	Touche relâchée
onLoad	Fin de chargement d'un élément
onMouseOver	Curseur de la souris au-dessus d'un élément
onMouseOut	Curseur de la souris qui s'éloigne d'un élément
onReset	Remise à zéro d'un formulaire
onResize	Redimensionnement du navigateur
onSelect	Sélection d'un texte
onSubmit	Soumission d'un formulaire
onUnload	Déchargement d'un élément

Événements (4)

Exemple :

```
<head>
  <title>Simple Clic</title>
  <meta charset="utf-8" />
  <script>
    function unClicSurUnElement()
    {
      alert("Un clic vient de se produire.");
    }
  </script>
</head>
<body>
  <p onClick="unClicSurUnElement();">Hello World</p>
</body>
```

Objets prédéfinis – *Number*

Permet d'utiliser les nombres comme des objets.

Utilisé principalement pour effectuer des conversions ou des manipulations sur les nombres.

Quelques exemples de propriétés et de méthodes :

`Number.MAX_VALUE` : La valeur numérique maximale qu'on peut représenter en JavaScript.

`Number.MIN_VALUE` : La plus petite valeur qu'on peut représenter en JavaScript, c'est-à-dire le plus petit nombre positif (le nombre le plus près de zéro qui n'est pas égal à zéro et qu'on peut représenter en JavaScript).

`Number.isNaN()` : Permet de déterminer si la valeur passée en argument vaut NaN.

`Number.isFinite()` : Permet de déterminer si la valeur numérique passée en argument est un nombre fini.

`Number.isInteger()` : Permet de déterminer si la valeur passée en argument est un entier.

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Number

Objets prédéfinis – *Math*

Permet l'utilisation de constantes et de fonctions mathématiques.

Quelques exemples de propriétés et de méthodes :

Math.E : Nombre d'Euler, la base des logarithmes naturels, environ 2,718..

Math.PI : Quotient de la circonférence d'un cercle par son diamètre, environ 3,14159.

...

Math.cos() : Renvoie le cosinus d'un nombre.

Math.sin() : Renvoie le sinus d'un nombre.

...

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Math

Objets prédéfinis – *Array* (1)

Déclaration :

```
var Tableau = new Array(12); OU var Tableau = [12, 14, 6, 4];
```

Exemple :

```
var Tableau = new Array();  
Tableau[0] = "debut"; Tableau[1] = "milieu"; Tableau[2] = "fin";  
console.log("La taille de ce tableau est : " + Tableau.length);
```

Tableaux multidimensionnels :

```
var matrice = new Array(2) ;  
Matrice[0] = new Array(2) ; matrice[1] = new Array(2) ;  
matrice[0][0] = 1 ; matrice[1][1] = 1 ;
```

Tableaux hétérogènes :

```
var t = [2, "lettre", true ]
```

Tableaux associatifs :

```
var t = new Array();  
t["Gaston"] = "Lagaffe";
```

Objets prédéfinis – *Array* (2)

Tableau = objet

Ex.

```
Tab = new Array()  
→ Tab['x'] = 1 OU Tab.x = 1
```

```
Obj = new Object  
→ Obj.x = 1 OU Obj['x'] = 1
```

Conséquence : parcours des propriétés d'un objet

```
indiv = new Individu("Martin", "rue Tabaga") ;  
for (var ppte in indiv)  
    {alert(ppte + "=" + indiv[ppte] + "\n");}
```

Objets prédéfinis – *Array* (3)

Propriété : length

Méthodes d'un objet array :

- **concat()** → concatène 2 tableaux : `tab1.concat(tab2)`
- **join()** → assemble les éléments dans une chaîne
EX : `var tab = ['A', 'B', 'C'];`
`tab.join()` → 'A, B, C'
- **pop()** → retire et retourne le dernier élément
- **shift()** → retire et retourne le premier élément
- **reverse()** → inverse l'ordre du tableau
- **sort()** → trie suivant l'ordre ASCII

Objets prédéfinis – *String*

Méthodes pour l'affichage en HTML:

`bold()` `italics()` `sub()` `sup()` `fontcolor()` `fontsize()`

Ex1: `texte.fontsize(10);`

Ex2: `texte.bold().fontsize(10);`

Méthodes pour la manipulation : Ex:

`citation = "Eureka";`

`citation.charAt(4) = k`

`citation.indexOf("ka") = 4`

`citation.indexOf("ki") = -1`

`citation.toLowerCase() = "eureka"`

`citation.substring(1,4) + citation.charAt(5) = ureka`

`citation.split("e") = { "ur", "ka" }`

Objets prédéfinis – *Date* (1)

Ex : October 29, 2004, 09:00:00

Création :

`today = new Date()` → date de l'instant d'exécution

`PremierMai2013 = new Date(2013, 4, 1, 9, 18, 37)`

`PremierMai2013 = new Date("may, 2013, 1")`

Méthodes :

Lecture d'éléments de la date :

Ex : `getDay()` → numéro du jour (dimanche=0, lundi=1, ...)

Modification d'éléments de la date :

Ex : `setHours(11)` → fixe l'heure à 11

Objets prédéfinis – *Date* (2)

Conversion pour l'affichage :

`toLocaleString()` `toGMTString()` `toUTCString` `toString`

Exemple :

```
d = new Date();  
alert(d + "\n");  
alert(d.toString());  
alert(d.toLocaleString());  
alert(d.toGMTString());  
alert(d.toUTCString());  
alert(d.getTime());
```

→ Fri Feb 15 2013 17:26:03 GMT+0100 (CET)

→ Fri Feb 15 2013 17:26:03 GMT+0100 (CET)

→ ven. 15 févr. 2013 17:26:03 CET

→ Fri, 15 Feb 2013 16:26:03 GMT

→ Fri, 15 Feb 2013 16:26:03 GMT

→ 1360945563127