Plan

Plan

JavaScript

Définition et caractéristiques

- est un langage de programmation de scripts orienté objet (à prototype)
- crée par Brendan Eich
- présenté par Netscape et Sun en décembre 1995
- standardisé par ECMAScript en juin 1997
- complète l'aspect algorithmique manquant à HTML (et CSS)

JavaScript

Spécificités du JavaScript

- langage faiblement typé
- syntaxe assez proche de celle de Java, C++, C...
- possibilité d'écrire plusieurs instructions sur une seule ligne à condition de les séparer par `;`
- terminer une instruction par `;` est fortement recommandée même si cette dernière est la seule sur la ligne

JavaScript

Autres langages de programmation de scripts

- AppleScript
- JScript, VBScript et TypeScript (Microsoft)
- LiveScript (Netscape) puis JavaScript
- ActionScript (MacroMedia)
- CoffeeScript (Open-source)
- ...

JavaScript

Trois façons pour définir des scripts JavaScript

- comme valeur d'attribut de n'importe quelle balise HTML
- dans une balise `<script>` de la section `<head>` d'une page HTML
- dans un fichier d'extension `.js` référencé dans une page HTML par la balise `<script>`

JavaScript

Première méthode :

```
<button onclick="alert('Hello World');"> Cliquer ici
 </button>
```

JavaScript

Deuxième méthode :

```
<!DOCTYPE html>
<html>
  <head>
    <title>First JS Page</title>
    <script type="text/javascript">
      function maFonction(){
        alert("Hello World !");
      }
    </script>
  </head>
  <body>
    <button onclick="maFonction()"> Cliquer ici</button>
  </body>
</html>
```

`type="text/javascript"` n'est plus nécessaire depuis HTML 5.

JavaScript

Troisième méthode

```
<!DOCTYPE html>
<html>
  <head>
    <title>First JS Page</title>
    <script  src="file.js"></script>
  </head>
  <body>
    <button onclick="maFonction()"> Cliquer ici</button>
  </body>
</html>
```

JavaScript

Troisième méthode

```
<!DOCTYPE html>
<html>
  <head>
    <title>First JS Page</title>
    <script  src="file.js"></script>
  </head>
  <body>
    <button onclick="maFonction()"> Cliquer ici</button>
  </body>
</html>
```

Contenu du `file.js`

```
function maFonction(){
    alert("Hello World !");
}
```

JavaScript

Autre box d'affichage avec confirmation

```
var bin = confirm("Press a button!");
alert(bin);
```

JavaScript

Autre box d'affichage avec confirmation

```
var bin = confirm("Press a button!");
alert(bin);
```

Autre box d'affichage avec une zone de saisie

```
var str = prompt("Votre nom", "John Wick");
alert(str);
```

JavaScript

La console, pourquoi ?

permet de contrôler l'avancement de l'exécution d'un programme (déboguer)

- en affichant le contenu de variables
- en vérifiant les blocs du code visités lors d'une exécution...

JavaScript

La console, pourquoi ?

permet de contrôler l'avancement de l'exécution d'un programme (déboguer)

- en affichant le contenu de variables
- en vérifiant les blocs du code visités lors d'une exécution...

Modifions le contenu du `file.js`

```
function maFonction(){
    console.log("Hello World !");
}
```

La console, comment ?

Où trouve t-on le message ?

- Pour les navigateurs suivants
  - **Google chrome**
  - **Mozilla firefox**
  - **Internet explorer**
- Cliquer sur F12

JavaScript

Existe t-il un autre moyen de tester un programme JS sans passer par un navigateur ?

- Oui, en utilisant NodeJS (pour télécharger `https://nodejs.org/en/`)
- Pour tester, utiliser une console telle que
  - Invite de commandes
  - Windows PowerShell
  - Cmder
- Lancer la commande node nomFichier.js

JavaScript

Modifions le contenu du `file.js`

```
function maFonction(){
    console.log("Hello World !");
}
maFonction();
```

JavaScript

Modifions le contenu du `file.js`

```
function maFonction(){
    console.log("Hello World !");
}
maFonction();
```

Lancer la commande `node file.js`

JavaScript

Il est aussi possible de définir un raccourci de `console.log`

```
var cl = console.log;
cl("Hello World !");
```

JavaScript

Commentaire sur une seule ligne

```
// commentaire
```

Commentaire sur une plusieurs lignes

```
/* le commentaire
   la suite
   et encore la suite
*/
```

Commentaire pour la documentation

```
/** un commentaire
   pour
   la documentation
*/
```

JavaScript

Déclaration d'une variable

```
var x;
```

JavaScript

Déclaration d'une variable

```
var x;
```

Initialisation

```
x = 0;
```

JavaScript

Déclaration d'une variable

```
var x;
```

Initialisation

```
x = 0;
```

Déclaration + initialisation

```
var y = 5 ;
```

JavaScript

Initialisation d'une variable non-déclarée $\Rightarrow$ déclaration

```
z = 5;
```

JavaScript

Initialisation d'une variable non-déclarée $\Rightarrow$ déclaration

```
z = 5;
```

Affectation d'une variable non-déclarée $\Rightarrow$ déclaration

```
t = x + y;
```

JavaScript

Utiliser une variable non-déclarée et non-initialisée $\Rightarrow$ erreur

```
alert (v);
```

JavaScript

Utiliser une variable non-déclarée et non-initialisée ⇒ erreur

```
alert (v);
```

Une variable déclarée mais non-initialisée a par défaut la valeur `undefined`

```
var w;
alert (w);
```

JavaScript

Utiliser une variable non-déclarée et non-initialisée ⇒ erreur

```
alert (v);
```

Une variable déclarée mais non-initialisée a par défaut la valeur `undefined`

```
var w;
alert (w);
```

`NaN` **: not a number**

```
var x = "bonjour";
n = x * 2;
console.log(n);
```

JavaScript

Les types de variables selon la valeur

- `number`
- `string`
- `boolean`
- `object`
- `undefined`

JavaScript

Récupérer le type de valeur affectée à une variable

```
console.log(typeof 5);
console.log(typeof true);
console.log(typeof 5.2);
console.log(typeof "bonjour");
console.log(typeof 'c');
var x;
console.log(typeof x);
console.log(typeof {nom:"wick", prenom:"john"});
console.log(typeof new Date());
console.log(typeof [1,2,3,4]);
console.log(typeof null);
```

JavaScript

Addition (qui peut se transformer en concaténation)

```
var x = 1;
y = 3;
z = '8';
t = "2";
u = "bonjour";
v = "3bonjour";
var m;
n = 2.5;
console.log(x + y);
console.log(x + z);
console.log(x + t);
console.log(x + y + z);
console.log(x + u);
console.log(x + v);
console.log(u + v);
console.log(x + m);
console.log(x + n);
```

JavaScript

Autres opérateurs arithmétiques

- $*$: multiplication
- $-$: soustraction
- $/$: division
- $\%$: reste de la division

JavaScript

Cependant, une division par 0 ne déclenche pas une exception

```
a = 2;
b = 0;

console.log(a / b);
```

JavaScript

Cependant, une division par 0 ne déclenche pas une exception

```
a = 2;
b = 0;

console.log(a / b);
```

Pour convertir une chaîne en entier

```
var a = '4';
b = 2;

console.log(b + parseInt(a));
```

JavaScript

Autres opérateurs arithmétiques

- `i++;` ≡ `i = i + 1;`
- `i--;` ≡ `i = i - 1;`
- `i += 2;` ≡ `i = i + 2;`
- `i -= 3;` ≡ `i = i - 3;`
- `i *= 2;` ≡ `i = i * 2;`
- `i /= 3;` ≡ `i = i / 3;`
- `i %= 5;` ≡ `i = i % 5;`

JavaScript

Pour permuter le contenu de deux variables

```
a = 2;
b = 0;

[a,b] = [b,a]
```

JavaScript

Pour permuter le contenu de deux variables

```
a = 2;
b = 0;

[a,b] = [b,a]
```

Pour évaluer une expression arithmétique exprimée sous forme de chaîne de caractères

```
var str = "2 + 5 * 3";

console.log(eval(str));
// affiche 17
```

JavaScript

Méthodes utiles pour les chaînes de caractères

- `length` : la longueur de la chaîne
- `toUpperCase()` : pour convertir une chaîne de caractères en majuscule
- `toLowerCase()` : pour convertir une chaîne de caractères en minuscule
- `trim()` : pour supprimer les espaces
- `substr()` : pour extraire une sous-chaîne de caractère
- `indexOf()` : pour retourner la position d'une sous-chaîne dans une chaîne, -1 sinon.
- ...

JavaScript

Pour accéder à un caractère d'indice `i` dans une chaîne de caractère

```
// soit directement via l'indice
console.log(str[i]));

// soit en faisant l'extraction d'une sous chaine de
  taille 1
console.log(str.substr(i,1));

// soit avec la methode d'extraction de caractere
console.log(str.charAt(i));
```

JavaScript

Tester une condition

```
if (condition1) {
    ...
}
[
else if (condition2) {
    ...
}
...
else {
    ...
}
]
```

JavaScript

Opérateurs logiques

- `&&` : et
- `||` : ou
- `!` : non

JavaScript

Opérateurs logiques

- `&&` : et
- `||` : ou
- `!` : non

Tester plusieurs conditions (en utilisant des opérateurs logiques)

```
if (condition1 && !condition2 || condition3){
    ...
}
[else ...]
```

JavaScript

Opérateurs logiques

- `&&` : et
- `||` : ou
- `!` : non

Tester plusieurs conditions (en utilisant des opérateurs logiques)

```
if (condition1 && !condition2 || condition3){
    ...
}
[else ...]
```

Pour les conditions, on utilise des opérateurs de comparaisons

JavaScript

Opérateurs de comparaison

- == : pour tester l'égalité des valeurs
- != : pour tester l'inégalité des valeurs
- === : pour tester l'égalité des valeurs et des types
- !== : pour tester l'inégalité des valeurs ou des types
- > : supérieur à
- < : inférieur à
- >= : supérieur ou égal à
- <= : inférieur ou égal à

JavaScript

Structure conditionnelle avec `switch`

```
var x = 5;
switch (x) {
    case 1:
        alert('un');
    break;

    case 2:
        alert('deux');
    break;

    case 3:
        alert('trois');
    break;

    default:
        alert("autre");
}
```

JavaScript

La variable dans switch peut être

- un nombre
- un caractère
- une chaîne de caractère (contrairement aux Java, C++, C...)

JavaScript

Considérons l'exemple suivant

```
var message = prompt('Saisir un message');

switch (message) {
    case "bonjour":
        alert('salut !');
    break;

    default:
        alert('quoi ?');
}
```

JavaScript

Considérons l'exemple suivant

```
var message = prompt('Saisir un message');

switch (message) {
    case "bonjour":
        alert('salut !');
    break;

    default:
        alert('quoi ?');
}
```

Simplifions l'écriture avec l'expression ternaire

```
var message = prompt('Saisir un message');

message == "bonjour" ? alert("salut !") : alert('quoi ?')
 ;
```

JavaScript

Boucle `while`

```
while (condition[s]) {
    ...
}
```

JavaScript

Boucle `while`

```
while (condition[s]) {
    ...
}
```

Boucle `do ... while`

```
do {
    ...
}
while (condition[s]);
```

JavaScript

Boucle `while`

```
while (condition[s]) {
    ...
}
```

Boucle `do ... while`

```
do {
    ...
}
while (condition[s]);
```

Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

JavaScript

Boucle `for`

```
for (initialisation; condition[s]; incrementation) {
    ...
}
```

JavaScript

Boucle `for`

```
for (initialisation; condition[s]; incrementation) {
    ...
}
```

Attention aux boucles infinies si vous modifiez la valeur du compteur à l'intérieur de la boucle.

JavaScript

Déclaration

```
var tab = new Array(value1, value2,... valueN);
```

JavaScript

Déclaration

```
var tab = new Array(value1, value2,... valueN);
```

Le raccourci

```
var tab = [value1, value2,... valueN];
```

Accès à un élément du tableau d'indice `i`

```
tab[i]
```

JavaScript

Opérations sur les tableaux

- `push` : pour ajouter un élément au tableau
- `pop()` : pour supprimer le dernier élément d'un tableau
- `shift()` : pour supprimer le premier élément du tableau
- `indexOf()` : pour retourner la position d'un élément dans un tableau, -1 sinon.
- `reverse()` : pour inverser l'ordre des élément d'un tableau
- `sort()` : pour trier un tableau
- `splice()` : pour extraire, ajouter ou supprimer un ou plusieurs éléments (selon les paramètres, voir page suivante)
- ...

JavaScript

Exemple avec `splice`

```
var sports = ["foot", "tennis", "basket", "volley"];

tab = sports.splice(2, 0, "rugby", "natation");

for(elt of sports)
        console.log(elt);
// affiche foot, tennis, rugby, natation, basket,
 volley

console.log(tab);
// n'affiche rien
```

Aucun élément supprimé car le deuxième paramètre = 0

JavaScript

Exemple avec `splice`

```
var sports = ["foot", "tennis", "basket", "volley"];

tab = sports.splice(2, 1, "rugby", "natation");

for(elt of sports)
        console.log(elt);
// affiche foot, tennis, rugby, natation,  volley

console.log(tab);
// affiche
```

Un seul élément supprimé car le deuxième paramètre = 1

JavaScript

À ne pas confondre avec `slice` qui permet d'extraire un sous-tableau sans modifier le tableau d'origine

```
var sports = ["foot", "tennis", "basket", "volley"];

tab = sports.slice(1, 3);

for(elt of sports)
        console.log(elt);
// affiche foot, tennis, basket,  volley

console.log(tab);
// affiche tennis, basket
```

JavaScript

Autres opérations sur les tableaux (ES5)

- `forEach()` : pour parcourir un tableau
- `map()` : pour appliquer une fonction sur les éléments d'un tableau
- `filter()` : pour filtrer les éléments d'un tableau selon un critère défini sous forme d'une fonction anonyme ou lambda
- `reduce()` : pour réduire tous les éléments d'un tableau en un seul selon une règle définie dans une fonction anonyme ou lambda
- `...`

JavaScript

Exemple avec `map`, `filter`, `forEach`

```
var tab =[2, 3, 5];
tab.map(x => x + 3)
    .filter(x => x > 5)
    .forEach(
        function(a,b){
            console.log(a-3);
        }
    );
    // affiche 3 et 5
```

JavaScript

Exemple avec `map`, `filter`, `forEach`

```
var tab =[2, 3, 5];
tab.map(x => x + 3)
    .filter(x => x > 5)
    .forEach(
        function(a,b){
            console.log(a-3);
        }
    );
    // affiche 3 et 5
```

On peut permuter `map` et `filter` selon le besoin

JavaScript

Exemple avec `map`, `filter`, `forEach`

```
var tab =[2, 3, 5];
tab.map(x => x + 3)
    .filter(x => x > 5)
    .forEach(
        function(a,b){
            console.log(a-3);
        }
    );
    // affiche 3 et 5
```

On peut permuter `map` et `filter` selon le besoin

On peut aussi remplacer les fonctions lambda par des fonctions anonymes

JavaScript

Exemple avec `reduce` : permet de réduire les éléments d'un tableau en une seule valeur

```
var tab =[2, 3, 5];
var somme = tab.map(x => x + 3)
    .filter(x => x > 5)
    .reduce(function(sum, elem){return sum + elem;});
console.log(somme);

// affiche 14
```

JavaScript

Parcourir un tableau avec une version simplifiée de la boucle `for`

```
var tab = [2, 3, 5];

for(elt of tab)
        console.log(elt);
```

JavaScript

Déclarer une fonction

```
function nomFonction([les arguments]){
  les instructions de la fonction
}
```

JavaScript

Déclarer une fonction

```
function nomFonction([les arguments]){
  les instructions de la fonction
}
```

Exemple

```
function somme(a, b){
        return a + b;
}
```

JavaScript

Déclarer une fonction

```
function nomFonction([les arguments]){
  les instructions de la fonction
}
```

Exemple

```
function somme(a, b){
        return a + b;
}
```

Appeler une fonction

```
var resultat = somme (1, 3);
```

JavaScript

Déclarer une fonction anonyme et l'affecter à une variable

```
var nomVariable = function ([les arguments]){
  les instructions de la fonction
}
```

JavaScript

Déclarer une fonction anonyme et l'affecter à une variable

```
var nomVariable = function ([les arguments]){
  les instructions de la fonction
}
```

Exemple

```
var somme2 = function (a, b){
        return a + b;
}
```

JavaScript

Déclarer une fonction anonyme et l'affecter à une variable

```
var nomVariable = function ([les arguments]){
  les instructions de la fonction
}
```

Exemple

```
var somme2 = function (a, b){
        return a + b;
}
```

Appeler une fonction anonyme

```
var resultat2 = somme2 (1, 3);
```

JavaScript

Déclarer une fonction en utilisant les expressions Lambda

```
var nomFonction = ([les arguments]) => {
  les instructions de la fonction
}
```

JavaScript

Déclarer une fonction en utilisant les expressions Lambda

```
var nomFonction = ([les arguments]) => {
  les instructions de la fonction
}
```

Exemple

```
var somme3 = (a,b) => { return a+b; }
```

JavaScript

Déclarer une fonction en utilisant les expressions Lambda

```
var nomFonction = ([les arguments]) => {
  les instructions de la fonction
}
```

Exemple

```
var somme3 = (a,b) => { return a+b; }
```

Appeler une fonction anonyme

```
var resultat3 = somme3 (1, 3);
```

JavaScript

Déclarer une fonction en utilisant un constructeur de fonction

```
var nomFunction = new Function (["param1", ... , "
 paramN",] "instructions");
```

JavaScript

Déclarer une fonction en utilisant un constructeur de fonction

```
var nomFunction = new Function (["param1", ... , "
 paramN",] "instructions");
```

Exemple

```
var somme4 = new Function ("a", "b", "return␣a␣+␣b")
 ;
```

JavaScript

Déclarer une fonction en utilisant un constructeur de fonction

```
var nomFunction = new Function (["param1", ... , "
 paramN",] "instructions");
```

Exemple

```
var somme4 = new Function ("a", "b", "return␣a␣+␣b")
 ;
```

Appeler une fonction anonyme

```
var resultat4 = somme4 (1, 3);
```

JavaScript

Et si on appelle la fonction `somme` sans passer des paramètres

```
console.log(somme());
```

JavaScript

Et si on appelle la fonction `somme` sans passer des paramètres

```
console.log(somme());
```

Le résultat

```
NaN
```

JavaScript

Et si on appelle la fonction `somme` sans passer des paramètres

```
console.log(somme());
```

Le résultat

```
NaN
```

On peut affecter une valeur par défaut aux paramètres

```
function somme (a=0, b=0){
        return a + b;
}
```

JavaScript

Et si on veut que la fonction `somme` retourne la somme quel que soit le nombre de paramètres passé

```
function somme (){
        result = 0;
        for(var i = 0; i < arguments.length ; i++){
                result += arguments[i];
        }
        return result;
}
```

JavaScript

Et si on veut que la fonction `somme` retourne la somme quel que soit le nombre de paramètres passé

```
function somme (){
        result = 0;
        for(var i = 0; i < arguments.length ; i++){
                result += arguments[i];
        }
        return result;
}
```

Tous ces appels sont corrects

```
console.log(somme(2,3,5));
console.log(somme(2,3));
console.log(somme(2,3,5,7));
console.log(somme(2));
```

JavaScript

Création d'un objet (ensemble de clé : valeur)

```
var obj = {nom:"wick", prenom:"john"};
```

JavaScript

Création d'un objet (ensemble de clé : valeur)

```
var obj = {nom:"wick", prenom:"john"};
```

Accès à un attribut de l'objet

```
console.log(obj.nom);
console.log(obj["prenom"]);
```

JavaScript

Création d'un objet (ensemble de clé : valeur)

```
var obj = {nom:"wick", prenom:"john"};
```

Accès à un attribut de l'objet

```
console.log(obj.nom);
console.log(obj["prenom"]);
```

Afficher les clés/valeurs d'un objet

```
for(key in obj)
    console.log(key + "␣:␣" + obj[key]);
```

JavaScript

On peut aussi faire

```
var obj = new Object();
obj.nom = "wick";
obj.prenom = "john";
```

JavaScript

On peut aussi faire

```
var obj = new Object();
obj.nom = "wick";
obj.prenom = "john";
```

Afficher les clés/valeurs d'un objet

```
for(key in obj)
    console.log(key + "␣:␣" + obj[key]);
```

JavaScript

Copier un objet

```
var obj2 = obj;
```

JavaScript

Copier un objet

```
var obj2 = obj;
```

Modifier un $\Rightarrow$ modifier l'autre

```
obj2.nom = "travolta";
console.log(obj.nom);
// affiche travolta
```

JavaScript

Copier un objet

```
var obj2 = obj;
```

Modifier un $\Rightarrow$ modifier l'autre

```
obj2.nom = "travolta";
console.log(obj.nom);
// affiche travolta
```

Ajouter un nouvel attribut à un objet existant

```
obj2.age = 35;
```

JavaScript

Copier un objet

```
var obj2 = obj;
```

Modifier un $\Rightarrow$ modifier l'autre

```
obj2.nom = "travolta";
console.log(obj.nom);
// affiche travolta
```

Ajouter un nouvel attribut à un objet existant

```
obj2.age = 35;
```

Supprimer un attribut d'un objet existant

```
delete obj.nom;
```

JavaScript

Pour cloner un objet

```
function clone(obj){
    var obj2 = {};
    for (key in obj)
        obj2[key] = obj[key];
    return obj2;
}
var obj = {nom: "wick", prenom: "john"};
var obj2 = clone(obj);

obj2.nom = "abruzzi";

console.log(obj);
console.log(obj2);
```

JavaScript

Ou tout simplement

```
var obj2 = Object.assign({}, obj);

console.log(obj);
console.log(obj2);
```

La méthode `Object.create()` permet de copier un objet.

JavaScript

Pour récupérer l'ensemble des valeurs d'un objet dans un tableau

```
var obj = {nom: "wick", prenom: "john"};

var values = Object.values(obj);

for (value of values)
    console.log(value); // affiche wick en suite
     john
```

JavaScript

Pour récupérer l'ensemble des clés d'un objet dans un tableau

```
var keys = Object.keys(obj);

for (key of keys)
    console.log(key + " : " + obj[key]);
// affiche nom : wick
// prenom : john
```

JavaScript

Pour transformer un objet en chaîne de caractère

```
var perso = { nom: 'wick', prenom: 'john' };
var str = JSON.stringify(perso);
console.log(str);
// affiche {"nom":"wick","prenom":"john"}
```

JavaScript

Pour transformer un objet en chaîne de caractère

```
var perso = { nom: 'wick', prenom: 'john' };
var str = JSON.stringify(perso);
console.log(str);
// affiche {"nom":"wick","prenom":"john"}
```

Pour transformer une chaîne de caractère en objet

```
var p = JSON.parse("str");
console.log(p.nom + "␣" + p.prenom);
// affiche wick john
```

JavaScript

Besoin d'un moule pour créer des objets (comme les classes en Java, C#, PHP...)

- Tous les objets JS sont de type `Object`
- Il est possible de créer ou de cloner un objet à partir d'un autre en utilisant des méthodes d'`Object`
- Et si on veut créer un modèle d'objet (comme la classe), on peut utiliser les constructeurs et les prototypes

JavaScript

Déclarer un constructeur d'objet

```
var Personne = function(nom,prenom){
        this.nom = nom;
        this.prenom = prenom;
}
```

JavaScript

Déclarer un constructeur d'objet

```
var Personne = function(nom,prenom){
        this.nom = nom;
        this.prenom = prenom;
}
```

Créer un objet

```
perso = new Personne("wick","john");
```

JavaScript

Déclarer un constructeur d'objet

```
var Personne = function(nom,prenom){
        this.nom = nom;
        this.prenom = prenom;
}
```

Créer un objet

```
perso = new Personne("wick","john");
```

Afficher les valeurs d'un objet

```
console.log(perso);
```

JavaScript

Une fois le constructeur déclaré, impossible de modifier le constructeur ainsi

```
Personne.age=0;
```

JavaScript

Une fois le constructeur déclaré, impossible de modifier le constructeur ainsi

```
Personne.age=0;
```

Pour cela, il faut utiliser le prototype

```
Personne.prototype.age=0;
```

JavaScript

Une fois le constructeur déclaré, impossible de modifier le constructeur ainsi

```
Personne.age=0;
```

Pour cela, il faut utiliser le prototype

```
Personne.prototype.age=0;
```

Pareillement pour les méthodes

```
Personne.prototype.nomComplet = function(){
        return this.nom + "␣" + this.prenom;
}
```

JavaScript

Une fois le constructeur déclaré, impossible de modifier le constructeur ainsi

```
Personne.age=0;
```

Pour cela, il faut utiliser le prototype

```
Personne.prototype.age=0;
```

Pareillement pour les méthodes

```
Personne.prototype.nomComplet = function(){
        return this.nom + "␣" + this.prenom;
}
```

Appeler la méthode ajoutée

```
perso.nomComplet();
```

JavaScript

Objets prédéfinis

- `Math` : contenant des méthodes permettant de réaliser des opérations mathématiques sur les nombres
- `Date` : permet de manipuler des dates
- ...

JavaScript

Exemple de méthodes de l'objet `Math`

```
Math.round(2.1);  // retourne 2
Math.round(2.9);  // retourne 3
Math.pow(2, 3);  // retourne 8
Math.sqrt(4);  // retourne 2
Math.abs(-2);  // retourne 2
Math.min(0, 1, 4, 2, -4, -5);  // retourne -5
Math.max(0, 1, 4, 2, -4, -5);  // retourne 4
Math.random();  // retourne un nombre aleatoire
Math.floor(Math.random() * 10); // retourne un
 entier compris entre 0 et 9
```

JavaScript

Créer et afficher un objet date

```
var date = new Date();
console.log(date); // affiche la date complete du jour
```

JavaScript

Créer et afficher un objet date

```
var date = new Date();
console.log(date); // affiche la date complete du jour
```

Créer une date à partir de valeurs passées en paramètre (les mois sont codés de 0 à 11)

```
var date = new Date(1985,7,30,5,50,0,0);
console.log(date); // affiche Fri Aug 30 1985 05:50:00
 GMT+0200
```

JavaScript

Créer et afficher un objet date

```
var date = new Date();
console.log(date); // affiche la date complete du jour
```

Créer une date à partir de valeurs passées en paramètre (les mois sont codés de 0 à 11)

```
var date = new Date(1985,7,30,5,50,0,0);
console.log(date); // affiche Fri Aug 30 1985 05:50:00
 GMT+0200
```

Créer une date à partir d'un nombre de millisecondes

```
var date = new Date(100000000000);
console.log(date); // affiche Sat Mar 03 1973 10:46:40
 GMT+0100
```

JavaScript

Exemple de méthodes de l'objet `Date`

- `getFullYear()` : retourne l'année
- `getMonth()` : retourne le mois (0-11)
- `getDate()` : retourne le jour (1-31)
- `getHours()` : retourne l'heure (0-23)
- `getMinutes()` : retourne les minutes (0-59)
- `getSeconds()` : retourne les secondes (0-59)
- `getMilliseconds()` : retourne les millisecondes (0-999)
- `getTime()` : retourne le nombre de millisecondes depuis le 1 Janvier 1970 (Avec ES5, on peut utiliser `Date.now()`)

On peut aussi modifier les attributs de l'objet `Date` en utilisant les `set`.

JavaScript

Expressions régulières

- outil de recherche puissant adopté par la plupart des langages de programmation
- facilitent la recherche dans les chaînes de caractères (et donc le remplacement, le calcul de nombre d'occurrences...)
- permettent de vérifier si des chaînes de caractères respectent certains formats (email, numéro de téléphone...)
- syntaxe plus ou moins proche pour tous les langages de programmation

JavaScript

Recherche d'une sous-chaîne dans une chaîne

```
var str = "Bonjour tout le monde";
var pos = str.search("tout");
console.log(pos); // affiche 8
```

JavaScript

Recherche d'une sous-chaîne dans une chaîne

```
var str = "Bonjour tout le monde";
var pos = str.search("tout");
console.log(pos); // affiche 8
```

Recherche avec une expression régulière insensible à la casse

```
var str = "Bonjour tout le monde";
var pos = str.search(/Tout/i);
console.log(pos); // affiche 8
```

JavaScript

Recherche d'une sous-chaîne dans une chaîne

```
var str = "Bonjour tout le monde";
var pos = str.search("tout");
console.log(pos); // affiche 8
```

Recherche avec une expression régulière insensible à la casse

```
var str = "Bonjour tout le monde";
var pos = str.search(/Tout/i);
console.log(pos); // affiche 8
```

Retourne -1 si la sous-chaîne n'existe pas

```
var str = "Bonjour tout le monde";
var pos = str.search(/c/i);
console.log(pos); // affiche -1
```

JavaScript

On peut aussi utiliser la fonction de recherche `test` qui retourne `true` si la sous-chaine existe, `false` sinon.

```
var str = /AB/i;
var result = str.test("ababaabbaaab");
console.log(result); // affiche true
```

JavaScript

Remplacer la première occurrence d'une sous-chaîne

```
var chaine = "ababaabbbaaab";
var txt = chaine.replace(/ab/,"c");
console.log(txt); // affiche cabaabbbaaab
```

JavaScript

Remplacer la première occurrence d'une sous-chaîne

```
var chaine = "ababaabbaaab";
var txt = chaine.replace(/ab/,"c");
console.log(txt); // affiche cabaabbaaab
```

Remplacer toutes les occurrences d'une sous-chaîne

```
var chaine = "ababaabbaaab";
var txt = chaine.replace(/ab/g,"c");
console.log(txt); // affiche ccacbaac
```

JavaScript

Contraintes exprimées avec les expressions régulières

- `+` : 1 ou plusieurs fois
- `*` : 0 ou plusieurs fois
- `?` : 0 ou 1 fois
- `{n,m}` : minimum `n` fois, maximum `m` fois
- `{n}` : `n` fois exactement
- `{n,}` : minimum `n` fois

JavaScript

Contraintes exprimées avec les expressions régulières

- `a|b` : a ou b
- `^` : commence par
- `$` : se termine par
- `()` : le groupe
- `a(?!b)` : a non suivi de b
- `a(?=b)` : a suivi de b

JavaScript

Contraintes exprimées avec les expressions régulières

- . : n'importe quel caractère
- $\backslash d$: un chiffre
- $\backslash D$: tout sauf un chiffre
- $\backslash w$: un caractère alphanumérique
- $\backslash W$: tout sauf un caractère alphanumérique
- $\backslash t$: un caractère de tabulation
- $\backslash n$: un caractère de retour à la ligne
- $\backslash s$: un espace
- ...

JavaScript

Contraintes exprimées avec les expressions régulières

- `[a-z]` : toutes les lettres entre a et z
- `[abcd]` : a, b, c, ou d
- `[A-Za-z]` : une lettre en majuscule et une en minuscule
- `[^a-d]` : tout sauf a, b, c, et d
- ...

Pour utiliser un caractère réservé (^, $...) dans une expression régulière, il faut le précéder par \

JavaScript

Exercice 1

Déterminer une expression régulière qui permet de déterminer si une chaîne de caractère contient 3 occurrences (pas forcément consécutives) de la sous-chaîne `ab`.

- `true` pour `abacccababcc`
- `true` pour `abababcccccab`
- `false` pour `baaaaabacccccba`

JavaScript

Exercice 2

Déterminer une expression régulière qui permet de déterminer si une chaîne commence par la lettre `a`, se termine par la lettre `b` et pour chaque `x` suivi d'un `y` (pas forcément consécutives), il existe un caractère `z` situé entre `x` et `y`.

- `true` pour `ab`
- `true` pour `abxyb`
- `true` pour `abxazyb`
- `false` pour `abxuyb`
- `false` pour `abxazyxyb`

JavaScript

Exercice 3

Déterminer une expression régulière qui permet de vérifier si une chaîne de caractère correspond à une adresse e-mail.

JavaScript

Exercice 3

Déterminer une expression régulière qui permet de vérifier si une chaîne de caractère correspond à une adresse e-mail.

Exercice 4

Déterminer une expression régulière qui permet de vérifier si une chaîne de caractère correspond à un numéro de téléphone.