

Chapitre 4 : JAVASCRIPT

I. INTRODUCTION AU COURS JAVASCRIPT

1. Introduction au JavaScript
2. L'environnement de travail pour ce cours JavaScript
3. Où écrire le code JavaScript ?
4. Commentaires, indentation et syntaxe de base en JavaScript

II. LES VARIABLES ET TYPES DE VALEURS JAVASCRIPT

- 1) Présentation des variables JavaScript
- 2) Les types de données en JavaScript
- 3) Présentation des opérateurs arithmétiques et d'affectation JavaScript
- 4) La concaténation et les littéraux de gabarits en JavaScript
- 5) Les constantes en JavaScript

III. LES STRUCTURES DE CONTRÔLE JAVASCRIPT

1. Structures de contrôle, conditions et opérateurs de comparaison JavaScript
2. Les conditions if, if...else et if...else if...else en JavaScript
3. Opérateurs logiques, précedence et règles d'associativité des opérateurs en JavaScript
4. Utiliser l'opérateur ternaire pour écrire des conditions JavaScript condensées
5. L'instruction switch en JavaScript
6. Présentation des boucles et des opérateurs d'incrément et de décrément en JavaScript
7. Les boucles while, do... while, for et for... in et les instructions break et continue en JavaScript

IV. LES FONCTIONS EN JAVASCRIPT

1. Présentation des fonctions JavaScript
2. Portée des variables et valeurs de retour des fonctions en JavaScript
3. Fonctions anonymes, auto-invoquées et récursives en JavaScript

V. L'ORIENTÉ OBJET EN JAVASCRIPT

- 1) Introduction à l'orienté objet en JavaScript
- 2) Création d'un objet JavaScript littéral et manipulation de ses membres
- 3) Définition et création d'un constructeur d'objets en JavaScript

- 4) Constructeur Object, prototype et héritage en JavaScript
 - 5) Les classes en JavaScript
-

VI. VALEURS PRIMITIVES ET OBJETS GLOBAUX JAVASCRIPT

- 1) Valeurs primitives et objets prédéfinis en JavaScript
 - 2) L'objet global JavaScript String, propriétés et méthodes
 - 3) L'objet global JavaScript Number, propriétés et méthodes
 - 4) L'objet global JavaScript Math, propriétés et méthodes
 - 5) Les tableaux en JavaScript et l'objet global Array
 - 6) Les dates en JavaScript et l'objet global Date
-

VII. MANIPULATION DU BOM EN JAVASCRIPT

- 1) JavaScript API, Browser Object Model et interface Window
 - 2) L'interface et l'objet Navigator et la géolocalisation en JavaScript
 - 3) L'interface et l'objet History en JavaScript
 - 4) L'interface et l'objet Location en JavaScript
 - 5) L'interface et l'objet Screen en JavaScript
-

VIII. MANIPULATION DU DOM EN JAVASCRIPT

- 1) Présentation du DOM HTML et de ses APIs accessibles en JavaScript
 - 2) Accéder aux éléments dans un document avec JavaScript et modifier leur contenu
 - 3) Naviguer ou se déplacer dans le DOM en JavaScript grâce aux noeuds
 - 4) Ajouter, modifier ou supprimer des éléments du DOM avec JavaScript
 - 5) Manipuler les attributs et les styles des éléments via le DOM en JavaScript
 - 6) La gestion d'évènements en JavaScript et la méthode addEventListener
 - 7) La propagation des évènements en JavaScript
 - 8) Empêcher un évènement de se propager et annuler son comportement par défaut en JavaScript
-

IX. UTILISATION DES EXPRESSIONS RÉGULIÈRES EN JAVASCRIPT

- 1) Introduction aux expressions régulières ou expressions rationnelles en JavaScript
- 2) Utiliser les expressions régulières pour effectuer des recherches et remplacements en JavaScript
- 3) Les classes de caractères et classes abrégées des expressions régulières JavaScript
- 4) Les métacaractères point, alternatives, ancres et quantificateurs des expressions régulières JavaScript

- 5) Créer des sous masques et des assertions dans les expressions régulières JavaScript
 - 6) Les drapeaux, options ou marqueurs des expressions régulières JavaScript
-

X. NOTIONS AVANCÉES SUR LES FONCTIONS JAVASCRIPT

1. Paramètres du reste et opérateur de décomposition des fonctions JavaScript
 2. Les fonctions fléchées JavaScript
 3. Les closures en JavaScript
 4. Gestion du délai d'exécution en JavaScript avec setTimeout() et setInterval()
-

XI. GESTION DES ERREURS ET MODE STRICT EN JAVASCRIPT

- 1) Gestion des erreurs en JavaScript
 - 2) Le mode strict en JavaScript
-

XII. L'ASYNCHRONE EN JAVASCRIPT

1. Introduction à l'asynchrone en JavaScript
 2. Les promesses en JavaScript
 3. Utiliser async et await pour créer des promesses plus lisibles en JavaScript
 4. Le chemin critique du rendu et les attributs HTML async et defer
-

XIII. SYMBOLES, ITÉRATEURS ET GÉNÉRATEURS EN JAVASCRIPT

- 1) Les symboles et l'objet Symbol en JavaScript
 - 2) Les protocoles et objets Iterable et Itérateur en JavaScript
 - 3) Les générateurs en Javascript
-

XIV. STOCKAGE DE DONNÉES DANS LE NAVIGATEUR EN JAVASCRIPT

- 1) Les cookies en JavaScript
 - 2) L'API Web Storage : localStorage et sessionStorage en JavaScript
 - 3) Utiliser l'API de stockage IndexedDB en JavaScript
-

XV. L'ÉLÉMENT HTML CANVAS ET L'API CANVAS

- 1) Présentation de l'élément HTML canvas et de l'API Canvas
- 2) Dessiner des rectangles dans un élément HTML canvas en Javascript
- 3) Définir des tracés pour dessiner des formes dans un canevas en JavaScript
- 4) Création de dégradés ou de motifs dans un canevas en JavaScript
- 5) Ajout d'ombres et utilisation de la transparence dans un canevas en JavaScript

- 6) Ajouter du texte ou une image dans un canevas en JavaScript
 - 7) Appliquer des transformations sur un canevas en JavaScript
-

xvi. LES MODULES JAVASCRIPT

- 1) Les modules JavaScript : import et export
-

xvii. JSON, AJAX ET FETCH EN JAVASCRIPT

- 1) Présentation de JSON et utilisation en JavaScript
 - 2) Introduction à l'Ajex en JavaScript
 - 3) Créer des requêtes Ajax en utilisant l'objet XMLHttpRequest en JavaScript
 - 4) Présentation et utilisation de l'API Fetch en Javascript
-

Objectifs du cours JavaScript et prérequis

Bienvenue dans ce cours complet traitant du langage de programmation JavaScript.

Ce cours aborde l'ensemble des fonctionnalités de base et utiles du JavaScript. L'objectif principal de celui-ci est de faire de vous des développeurs autonomes pour que vous puissiez résoudre par vous même la majorité des problématiques liées au JavaScript et que vous soyez capables de terminer des projets plus ou moins complexes.

L'idée derrière ce cours n'est donc pas simplement de vous présenter les différentes notions liées au JavaScript une à une mais bien que vous les compreniez et que vous sachiez quand et comment utiliser chacune d'entre elles dans le futur.

Pour cela, le cours est progressif dans sa difficulté au sein de chaque partie et entre les parties : nous allons commencer par définir les notions de base du JavaScript avec les variables, fonctions et autres structures de contrôle comme les boucles et les conditions et irons progressivement vers du plus complexe et du plus structuré avec l'orienté objet et l'utilisation d'APIs (Interfaces de Programmation) par exemple.

Cette progressivité rend le cours ouvert et accessible à tous : que vous ayez déjà des bases ou non en JavaScript, vous trouverez forcément quelque chose à en retirer.

Cependant, pour suivre ce cours dans de bonnes conditions, il est essentiel que vous possédiez des bases en HTML et en CSS. Si vous ne connaissez pas du tout ces deux langages, je vous invite à consulter mon cours complet traitant de ce sujet en cliquant [ici](#)

Pédagogie et méthodologie du cours JavaScript

Pour ce cours JavaScript, nous allons déjà commencer par définir le rôle du JavaScript et par vous créer un socle de connaissances solide pour vous permettre d'avoir une vue d'ensemble du langage.

Ensuite, nous irons le plus rapidement vers des exemples et exercices concrets pour valider l'acquisition de ces premières connaissances et pour être sûr de bien comprendre quand et comment utiliser chacune d'entre elles.

Chaque notion du cours est illustré par des exemples les plus concrets possibles. En effet, pour rendre le cours le plus digérable, intéressant, pédagogique et pratique possible, un fort accent a été mis sur la pratique.

Dans un but pédagogique, et afin de vous fournir la meilleure expérience d'apprentissage possible, ce cours a été divisé en de multiples sous-chapitres eux mêmes regroupés en sections.

Vous pouvez à tout moment voir le sommaire du cours sur votre gauche pour vous repérer et naviguer entre les différentes leçons du cours.

Les codes des différents exemples vous seront fournis. Cependant, je vous conseille d'adopter une attitude active et de vous exercer sur chacun d'entre eux plutôt que de simplement les copier-coller si vous souhaitez véritablement progresser : je vous rappelle ici qu'on n'apprend vraiment à coder qu'en pratiquant.

Dans cette première leçon d'introduction, nous allons définir ce qu'est le JavaScript ainsi que les principes fondateurs de ce langage et allons comprendre la place du JavaScript parmi les autres langages et ses usages.

I- Une première définition du JavaScript

Le JavaScript est un langage de programmation créé en 1995. Le JavaScript est aujourd'hui l'un des langages de programmation les plus populaires et il fait partie des langages web dits « standards » avec le HTML et le CSS. Son évolution est gérée par le groupe ECMA International qui se charge de publier les standards de ce langage.

On dit que le HTML, le CSS et le JavaScript sont des standards du web car les principaux navigateurs web (Google Chrome, Safari, Firefox, etc.) savent tous « lire » (ou « comprendre » ou « interpréter ») ces langages et les interprètent généralement de la même façon ce qui signifie qu'un même code va généralement produire le même résultat dans chaque navigateur.

Pour définir ce qu'est le JavaScript et le situer par rapport aux autres langages, et donc pour comprendre les intérêts et usages du JavaScript il faut savoir que :

- Le JavaScript est un langage dynamique ;
- Le JavaScript est un langage (principalement) côté client ;
- Le JavaScript est un langage interprété ;
- Le JavaScript est un langage orienté objet.

Pas d'inquiétude, on va définir le plus simplement possible ce que ces qualificatifs signifient !

a) Le JavaScript, un langage dynamique

Le JavaScript est un langage dynamique, c'est-à-dire un langage qui va nous permettre de générer du contenu dynamique pour nos pages web.

Un contenu « dynamique » est un contenu qui va se mettre à jour dynamiquement, c'est-à-dire changer sans qu'on ait besoin de modifier le code manuellement mais plutôt en fonction de différents facteurs externes.

On oppose généralement les langages « dynamiques » aux langages « statiques » comme le HTML et le CSS. Illustrons les différences d'utilisation entre ces types de langage en discutant des possibilités du HTML, du CSS et du JavaScript.

Pour rappel, le HTML est un langage de balisage (langage qui utilise des balises) qui est utilisé pour structurer et donner du sens aux différents contenus d'une page. Le HTML nous permet de communiquer avec un navigateur en lui indiquant que tel contenu est un titre, tel contenu est un simple paragraphe, tel autre est une liste, une image, etc.

Le navigateur comprend les différentes balises HTML et va alors afficher notre page à nos visiteurs en tenant compte de celles-ci.

Le contenu HTML ne va jamais être affiché tel quel, brut, mais des règles de mises en forme vont lui être appliquées. Ces règles de styles vont être définies en CSS. Le CSS va ainsi nous permettre d'arranger les différents contenus HTML de la page en les positionnant les uns par rapport aux autres, en modifiant la couleur des textes, la couleur de fond des éléments HTML, etc.

Le CSS va ainsi se charger de l'aspect visuel de notre page tandis que le HTML se charge de la structure (définir les contenus) de celle-ci.

Le HTML et le CSS forment ainsi un premier couple très puissant. Cependant, nous allons être limités si nous n'utilisons que ces deux langages tout simplement car ce sont des langages qui ne permettent que de créer des pages « statiques ».

Une page statique est une page dont le contenu est le même pour tout le monde, à tout moment. En effet ni le HTML ni le CSS ne nous permettent de créer des contenus qui vont se mettre à jour par eux-mêmes. Le CSS, avec les animations, nous permet de créer des styles pseudo-dynamiques mais tout de même prédéfinis.

C'est là où le JavaScript entre en jeu : ce langage va nous permettre de manipuler des contenus HTML ou des styles CSS et de les modifier en fonction de divers événements ou variables. Un événement peut être par exemple un clic d'un utilisateur à un certain endroit de la page tandis qu'une variable peut être l'heure de la journée.

Regardez par exemple le code suivant :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1,
user-scalable=no">
    <link rel="stylesheet" href="cours.css">
    <script>
      window.addEventListener('load',horloge);
      function horloge(){
        let d = new Date();
        document.getElementById('heure').innerHTML =
d.toLocaleTimeString();
        setTimeout(horloge, 1000);
      }

      document.addEventListener('DOMContentLoaded',
function(){
        let cache = document.getElementById('bouton');
        cache.addEventListener('click',cacheHorloge);
        document.getElementById('tog').style.display =
'block';
        function cacheHorloge(){
          let para = document.getElementById('tog');
          if(para.style.display == 'block'){
            para.style.display = 'none';
          }else{
            para.style.display = 'block';
          }
        }
      });
    </script>
  </head>
```

Titre principal

Il est actuellement 16:30:41

Cacher / Afficher l'heure

L'idée n'est bien sûr pas ici de vous expliquer comment fonctionne ce code qui est déjà relativement complexe mais de vous donner une idée de ce qu'on va pouvoir réaliser avec quelques lignes de JavaScript.

Mon code JavaScript est ici placé dans l'élément `head` de mon fichier HTML à l'intérieur d'un élément `script`. Ce code récupère l'heure actuelle et l'actualise toutes les secondes d'une part, et nous permet de cacher / d'afficher l'heure via un bouton d'autre part.

Ces deux fonctionnalités sont des fonctionnalités dynamiques qu'on n'aurait pas pu réaliser en HTML ni en CSS.

• Le JavaScript, un langage (principalement) côté client

La catégorisation langages statiques / langage dynamique est une première façon de classer les différents langages de programmation.

On peut également classer les différents langages selon l'endroit où ils vont s'exécuter : soit côté client, soit côté serveur.

Pour comprendre ce que sont les langages « côté client » et « côté serveur », il convient avant tout de comprendre ce qu'est un client et ce qu'est un serveur et pour cela il faut savoir ce qu'est un site.

Un site est un ensemble de ressources et de fichiers liés entre eux. Pour que notre site soit accessible sur le web pour tous, on va l'héberger sur un serveur, c'est-à-dire envoyer l'ensemble de nos fichiers sur le serveur et on va également acheter un nom de domaine qui va servir à identifier notre site.

Un « serveur » est une sorte de super ordinateur, constamment accessible et connectés aux autres serveurs (formant ainsi un réseau qu'on appelle le web) et qui va héberger les fichiers constituant un (ou plusieurs) site(s) web et le(s) « servir » sur demande du client.

Lorsqu'on demande à accéder à une page web en tapant une URL dans notre navigateur, nous sommes le client ou plus exactement notre navigateur est le logiciel client qui effectue une demande ou « requête » au serveur qui est la suivante : « sers-moi le fichier correspondant à l'adresse que je t'ai envoyée ».

Les fichiers ou pages d'un site web vont pouvoir être constituées de deux types de codes différents : du code côté serveur et du code côté client. Lorsqu'on demande à un serveur de nous servir une page, celui-ci se charge d'exécuter le code côté client s'il y en a et ne va renvoyer que du code côté client en résultat.

Un langage « côté client » ou « client side » est un langage qui va être exécuté dans le navigateur des utilisateurs qui demandent la page. On peut également appeler ces langages des langages « web » puisqu'ils sont principalement utilisés dans un contexte web.

Il existe aujourd'hui 3 langages côté client incontournables qui sont le HTML, le CSS et le JavaScript.

Les langages côté serveur sont des langages qui vont s'exécuter sur le serveur. Les navigateurs ne sont dans la grande majorité des cas pas capables de comprendre les langages serveur.

Ces langages permettent notamment d'effectuer de manipuler les données pour renvoyer des résultats. Les résultats renvoyés le sont sous forme de code compréhensible par le navigateur (c'est-à-dire du HTML principalement) pour que le navigateur puisse afficher le résultat final.

La chose importante à retenir ici est que le JavaScript est un langage principalement utilisé côté client, mais qui va également pouvoir s'utiliser côté serveur à condition qu'on mette en place un environnement favorable (en utilisant Node.js par exemple).

Dans ce cours, nous allons exclusivement nous concentrer sur un usage du JavaScript côté client.

- Le JavaScript, un langage interprété

On peut encore séparer les langages selon qu'ils puissent être exécutés directement (on parlera alors de langages interprétés) ou qu'il faille les transformer en une autre forme pour pouvoir les exécuter (on parlera alors de langages compilés).

Le JavaScript est un langage interprété. Cela signifie qu'il va pouvoir être exécuté directement sous réserve qu'on possède le logiciel interpréteur. Pas de panique ici : tous les navigateurs connus possèdent leur interpréteur JavaScript.

- Le JavaScript, un langage orienté objet

Finalement, le JavaScript est un langage orienté objet. Il est trop tôt selon moi pour vous expliquer ce que ça signifie ; nous reparlerons de cela dans la partie consacrée aux objets.

b) JavaScript, API, librairies et Framework

Le JavaScript en tant que langage correspond à un ensemble de structures de codes ou un ensemble d'éléments qu'on va pouvoir utiliser pour implémenter des fonctionnalités sur nos pages web.

Les API et les librairies JavaScript sont construites à partir de ces éléments de base du JavaScript et vont nous permettre d'utiliser des structures plus complexes déjà prêtes à l'emploi qui vont in-fine nous permettre de réaliser simplement des opérations qu'il aurait été très difficile de réaliser si on avait dû les coder entièrement à la main.

Une API (« Application Programming Interface » ou « Interface de Programmation ») est une interface qui nous permet d'utiliser facilement une application. Une application est un programme, c'est-à-dire un ensemble cohérent de code qui permet de réaliser certaines actions.

On utilise les API pour demander au programme d'effectuer certaines actions pour nous, comme par exemple afficher une carte d'une certaine ville à une certaine échelle (Google Maps API) ou pour afficher la liste de nos derniers Tweets (Twitter API) ou encore pour manipuler le contenu HTML d'une page web (DOM API).

Pour utiliser une API et donc l'application correspondante, il faudra généralement demander au propriétaire de l'application une clef qui va nous permettre de nous identifier.

Une librairie ou « bibliothèque » JavaScript est un ensemble de fichiers de code JavaScript homogènes (= qui se concentrent sur un aspect particulier du langage) qu'on va devoir télécharger pour les utiliser. Ces fichiers de code contiennent des structures de code prêtes à l'emploi qu'on va pouvoir utiliser immédiatement pour gagner du temps en développement. Parmi les librairies les plus célèbres, on peut notamment citer jQuery.

Il convient donc de ne pas confondre API et librairies : une librairie est un ensemble de fichiers qu'on va télécharger et contient un ensemble de structures de codes prêtes à l'emploi. Nous allons pouvoir choisir celles qui nous intéressent pour les intégrer dans nos propres scripts et ainsi gagner du temps de développement. Une API, de l'autre côté, va nous permettre d'utiliser une application qu'on n'a pas le droit de manipuler directement.

Finalement, un framework ou « cadre de travail » est relativement similaire dans son but à une « super librairie ». Les framework vont également nous fournir un ensemble de codes tout prêts pour nous faire gagner du temps en développement. La grande différence entre un framework et une librairie réside dans l'inversion du contrôle : lorsqu'on télécharge une

librairie, on peut l'utiliser comme on le souhaite en intégrant ses éléments à nos scripts tandis que pour utiliser un framework il faut respecter son cadre (ses règles). Les framework JavaScript les plus connus aujourd'hui sont Angular.js et React.js.

Dans le début de ce cours, nous n'utiliserons bien évidemment pas d'API ni de librairie et encore moins de framework. Cependant, il reste intéressant de déjà définir ces différents termes pour vous donner une première « vue d'ensemble » des outils JavaScript.

c) JavaScript vs Java : attention aux confusions !

Encore aujourd'hui, certaines personnes ont tendance à confondre les deux langages « Java » et « JavaScript ».

Retenez ici que ces deux langages, bien que syntaxiquement assez proches à la base, reposent sur des concepts fondamentaux complètement différents et servent à effectuer des tâches totalement différentes.

Pourquoi des noms aussi proches ? Java est une technologie créée originellement par Sun Microsystems tandis que JavaScript est un langage créé par la société Netscape.

Avant sa sortie officielle, le nom original du JavaScript était « LiveScript ». Quelques jours avant la sortie du LiveScript, le langage est renommé JavaScript.

A l'époque, Sun et Netscape étaient partenaires et le Java était de plus en plus populaire. Il est donc communément admis que le nom « JavaScript » a été choisi pour des raisons marketing et pour créer une association dans la tête des gens avec le Java afin que les deux langages se servent mutuellement.

Le créateur du JavaScript a également expliqué que l'idée de base derrière le développement du JavaScript était d'en faire un langage complémentaire au Java.

2) L'environnement de travail pour ce cours JavaScript

Pour coder en JavaScript, nous n'allons avoir besoin que d'un éditeur de texte. Il existe de nombreux éditeurs de texte sur le web et la majorité d'entre eux sont gratuits.

Si vous suivez ce cours, vous devriez déjà avoir des bases en HTML et en CSS et donc non seulement savoir ce qu'est un éditeur de texte mais en avoir déjà un installé sur votre ordinateur et prêt à l'utilisation.

Si jamais ce n'était pas le cas, je ne saurais que trop vous conseiller de suivre le cours HTML et CSS avant d'aller plus loin dans celui-ci.

Pour rappel, voici une courte liste d'éditeurs reconnus et qui vous permettront de coder en JavaScript sans problème (j'utilise à titre personnel la version gratuite de Komodo).

- Komodo Edit : version gratuite de Komodo, éditeur multiplateformes (il fonctionne aussi bien sous Windows que Mac ou encore Ubuntu). L'éditeur est complet, performant et relativement intuitif.
- Atom : Atom est doté d'une excellente ergonomie qui facilite grandement la prise en main et l'approche du code pour les nouveaux développeurs. Cet éditeur de texte dispose de toutes les fonctions qu'on attend d'un bon éditeur : bibliothèques intégrées, auto-complétion des balises, etc.
- Notepad++ : Certainement l'éditeur de texte le plus connu de tous les temps, Notepad++ est également l'un des plus anciens. Il a passé le test du temps et a su s'adapter au fur et à mesure en ajoutant des fonctionnalités régulièrement comme l'auto-complétion des balises, le surlignage des erreurs de syntaxe dans le code etc. Le seul bémol selon moi reste son interface qui est à peaufiner.
- Brackets : Brackets est un éditeur très particulier puisqu'il est tourné uniquement vers les langages de développement front-end (c'est-à-dire HTML, CSS et JavaScript). Cependant, il dispose d'une excellente ergonomie (UI / UX) et d'un support extensif pour les langages supportés.

2) Logiciel éditeur de texte contre éditeur en ligne

Certains sites comme codepen.io ou jsbin.com permettent d'écrire du code HTML, CSS ou JavaScript et de voir le résultat immédiatement.

En cela, ils servent le même rôle qu'un éditeur de texte mais sont encore plus pratiques, notamment lorsque vous voulez tester rapidement un bout de code ou pour des démonstrations de cours. J'utiliserai d'ailleurs souvent codepen.io dans mes leçons pour vous fournir directement un code et pour que vous puissiez le voir immédiatement en action ou le modifier.

Cependant, retenez bien qu'ils sont aussi limités car il y a plusieurs choses que vous ne pourrez pas faire en termes de développement avec ces sites. Parmi celles-ci, on notera que vous ne pourrez par exemple pas exécuter de code PHP ou un quelconque code utilisant un langage dit server side ou encore que vous ne pourrez pas à proprement parler créer plusieurs pages et les lier entre elles (comme c'est le cas lorsque l'on doit créer un site) ou du moins pas gratuitement.

Cette solution n'est donc pas satisfaisante si vous souhaitez véritablement vous lancer dans le développement et c'est la raison pour laquelle tous les développeurs utilisent un éditeur de texte.

Je vous conseille donc durant ce cours d'utiliser un maximum votre éditeur pour bien vous familiariser avec celui-ci et pour assimiler les différentes syntaxes des langages que l'on va étudier plutôt que de simplement copier et / ou utiliser mes codes que je mettrai à disposition via CodePen.

b) Les librairies JavaScript à télécharger

Pour coder en JavaScript, un simple éditeur de texte suffit en théorie. Cependant, pour exploiter toute la puissance du JavaScript et pour gagner du temps de développement, nous utiliserons généralement en plus du JavaScript « de base » des librairies JavaScript.

Pour qu'une librairie JavaScript fonctionne, il va falloir que le navigateur des personnes qui affichent la page la connaisse. Pour cela, on « forcera » le navigateur de nos visiteurs à télécharger les librairies qu'on utilise dans nos pages. Pour commencer, nous n'utiliserons pas de librairie. Nous aurons donc l'occasion de reparler de cela bien plus tard dans ce cours.

. Où écrire le code JavaScript ?

On va pouvoir placer du code JavaScript à trois endroits différents :

- Directement dans la balise ouvrante d'un élément HTML ;
- Dans un élément `script`, au sein d'une page HTML ;

- Dans un fichier séparé contenant exclusivement du JavaScript et portant l'extension `.js`.

Nous allons dans cette leçon voir comment écrire du code JavaScript dans chacun de ces emplacements et souligner les différents avantages et inconvénients liés à chaque façon de faire.

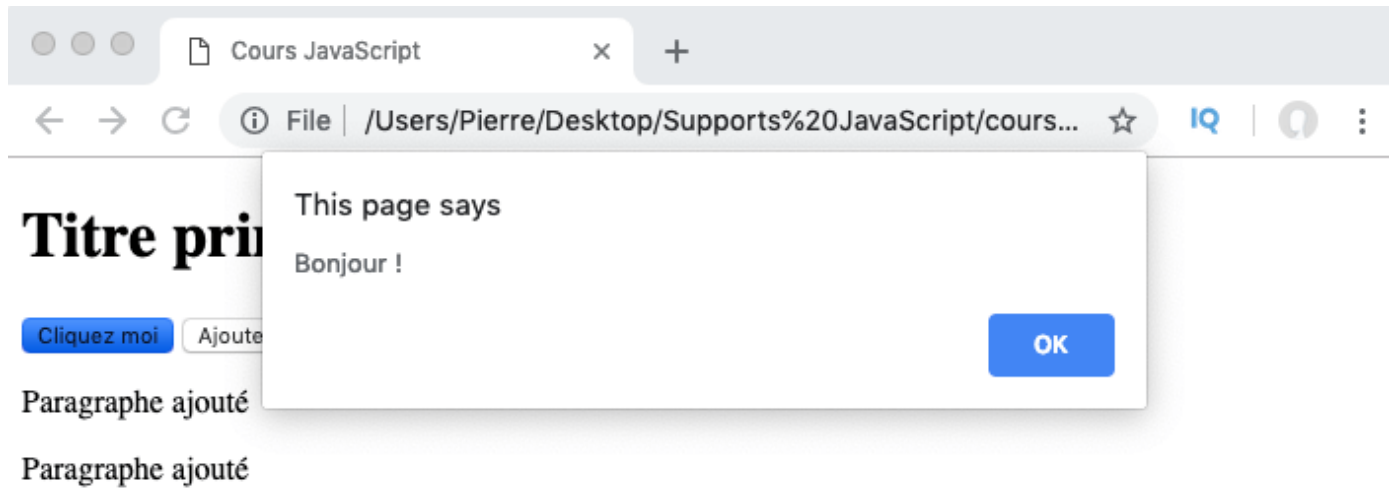
3) Placer le code JavaScript dans la balise ouvrante d'un élément HTML

Il est possible que vous rencontriez encore aujourd'hui du code JavaScript placé directement dans la balise ouvrante d'éléments HTML. Ce type de construction était fréquent à l'époque notamment pour prendre en charge des événements comme par exemple un clic. Regardez plutôt l'exemple ci-dessous :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, user-scalable=no">
    <link rel="stylesheet" href="cours.css">
  </head>

  <body>
    <h1>Titre principal</h1>
    <button onclick="alert('Bonjour !')">Cliquez moi</button>

    <button onclick="(function(){
      let para = document.createElement('p');
      para.textContent = 'Paragraphe ajouté';
      document.body.appendChild(para);
    })();">
      Ajouter un paragraphe
    </button>
  </body>
</html>
```

Ici, on crée deux boutons en HTML et on place nos codes JavaScript à l'intérieur d'attributs **onclick**. Le code à l'intérieur des attributs va s'exécuter dès qu'on va cliquer sur le bouton correspondant.

Dans le cas présent, cliquer sur le premier bouton a pour effet d'ouvrir une fenêtre d'alerte qui affiche « Bonjour ! ».

Cliquer sur le deuxième bouton rajoute un élément **p** qui contient le texte « Paragraphe ajouté » à la suite des boutons.

Je vous demande pour le moment de ne pas trop vous attarder sur les codes JavaScript en eux-mêmes car nous aurons largement le temps de découvrir les structures de ce langage dans la suite de ce cours mais plutôt de vous concentrer sur le sujet de la leçon qui concerne les emplacements possibles du code JavaScript.

Aujourd'hui, de nouvelles techniques nous permettent de ne plus utiliser ce genre de syntaxe et il est généralement déconseillé et considéré comme une mauvaise pratique d'écrire du code JavaScript dans des balises ouvrantes d'éléments HTML.

La raison principale à cela est que le web et les éléments le composant sont de plus en plus complexes et nous devons donc être de plus en plus rigoureux pour exploiter cette complexité.

Ainsi, la séparation des différents langages ou codes est aujourd'hui la norme pour essayer de conserver un ensemble le plus propre, le plus compréhensible et le plus facilement maintenable possible.

En plus de cela, polluer le code HTML comme cela peut conduire à certains bogues dans le code et est inefficace puisqu'on aurait à recopier les différents codes pour chaque élément auquel on voudrait les appliquer.

b) Placer le code JavaScript dans un élément script, au sein d'une page HTML

On va également pouvoir placer notre code JavaScript dans un élément `script` qui est l'élément utilisé pour indiquer qu'on code en JavaScript.

On va pouvoir placer notre élément `script` n'importe où dans notre page HTML, aussi bien dans l'élément `head` qu'au sein de l'élément `body`.

De plus, on va pouvoir indiquer plusieurs éléments `script` dans une page HTML pour placer plusieurs bouts de code JavaScript à différents endroits de la page.

Regardez plutôt l'exemple ci-dessous. Ce code produit le même résultat que le précédent :

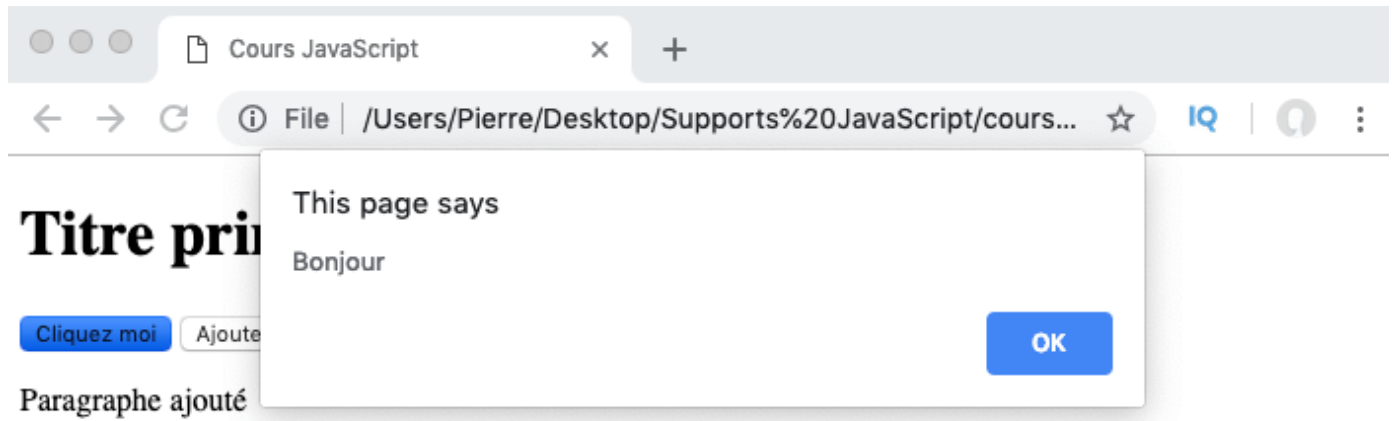
```

<!DOCTYPE html>
<html>
  <head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, user-scalable=no">
    <link rel="stylesheet" href="cours.css">
    <script>
      document.addEventListener('DOMContentLoaded', function(){
        let bonjour = document.getElementById('b1');
        bonjour.addEventListener('click', alerte);

        function alerte(){
          alert('Bonjour');
        }
      });
    </script>
  </head>

  <body>
    <h1>Titre principal</h1>
    <button id='b1'>Cliquez moi</button>
    <button id='b2'>Ajouter un paragraphe</button>
    <script>
      let ajouter = document.getElementById('b2');
      ajouter.addEventListener('click', ajout);
      function ajout(){
        let para = document.createElement('p');
        para.textContent = 'Paragraphe ajouté';
        document.body.appendChild(para);
      }
    </script>
  </body>
</html>

```



Cette méthode est meilleure que la précédente mais n'est une nouvelle fois pas idéalement celle que nous allons utiliser pour plusieurs raisons.

Tout d'abord, comme précédemment, la séparation des codes n'est pas optimale ici puisqu'on mélange du JavaScript et du HTML ce qui peut rendre l'ensemble confus et complexe à comprendre dans le cadre d'un gros projet.

De plus, si on souhaite utiliser les mêmes codes sur plusieurs pages, il faudra les copier-coller à chaque fois ce qui n'est vraiment pas efficient et ce qui est très mauvais pour la maintenabilité d'un site puisque si on doit changer une chose dans un code copié-collé dans 100 pages de notre site un jour, il faudra effectuer la modification dans chacune des pages.

. Placer le code JavaScript dans un fichier séparé

Placer le code JavaScript dans un fichier séparé ne contenant que du code JavaScript est la méthode recommandée et que nous préférons tant que possible.

Pour faire cela, nous allons devoir créer un nouveau fichier et l'enregistrer avec une extension `.js`. Ensuite, nous allons faire appel à notre fichier JavaScript depuis notre fichier HTML.

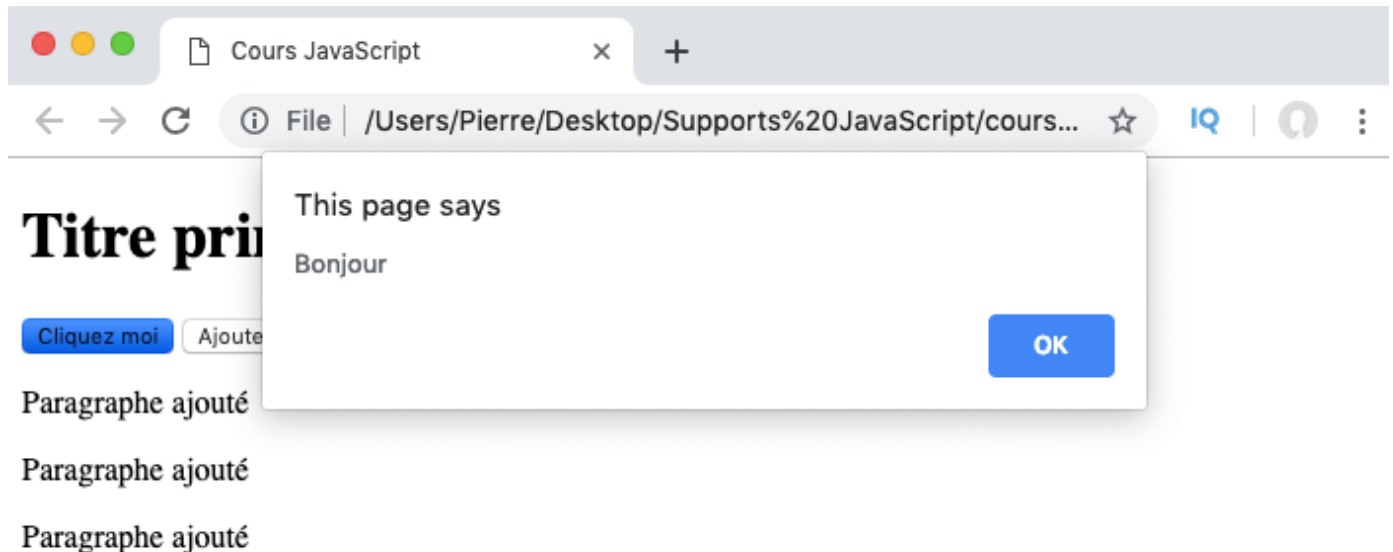
Pour cela, on va à nouveau utiliser un élément `script` mais nous n'allons cette fois-ci rien écrire à l'intérieur. A la place, on va plutôt ajouter un attribut `src` à notre élément `script` et lui passer en valeur l'adresse du fichier. Si votre fichier `.js` se situe dans le même dossier que votre fichier `.html`, il suffira d'indiquer le nom du fichier en valeur de l'attribut `src`.

Notez qu'un élément `script` ne peut posséder qu'un attribut `src`. Dans le cas où on souhaite utiliser plusieurs fichiers JavaScript dans un fichier HTML, il faudra renseigner autant d'éléments `script` dans le fichier avec chaque élément appelant un fichier en particulier.

Le code ci-dessous produit à nouveau les mêmes résultats que précédemment. Ne vous préoccupez pas de l'attribut `async` pour le moment.

```
cours.html x cours.js x cours.css x
... Pierre > Desktop > Supports JavaScript > cours.html > Ln: 19 Col: 1 UTF-8
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Cours JavaScript</title>
5     <meta charset="utf-8">
6     <meta name="viewport"
7       content="width=device-width, initial-scale=1, user-scalable=no">
8     <link rel="stylesheet" href="cours.css">
9     <script src='cours.js' async></script>
10  </head>
11
12  <body>
13    <h1>Titre principal</h1>
14    <button id='b1'>Cliquez moi</button>
15    <button id='b2'>Ajouter un paragraphe</button>
16  </body>
17 </html>
```

```
cours.html x cours.js x cours.css x
... Cours > JavaScript > Supports JavaScript > cours.js > Ln: 23 Col: 1 UTF-8
1 let bonjour = document.getElementById('b1');
2 let ajouter = document.getElementById('b2');
3
4 bonjour.addEventListener('click', alerte);
5 ajouter.addEventListener('click', ajout);
6
7 function alerte(){
8   alert('Bonjour');
9 }
10 function ajout(){
11   let para = document.createElement('p');
12   para.textContent = 'Paragraphe ajouté';
13   document.body.appendChild(para);
14 }
```



Cette méthode sera notre méthode préférée puisqu'elle permet une excellente séparation du code et une maintenabilité optimale de celui-ci. En effet, si on veut insérer le code JavaScript contenu dans notre fichier dans 100 pages différentes, il suffira ici d'appeler ce fichier JavaScript dans les 100 pages. En cas de modification du code, il suffira alors de le modifier une fois dans le fichier JavaScript.

Note : Pour illustrer la séparation des codes dans CodePen, je placerai désormais le code JavaScript dans l'onglet « JavaScript » de ce logiciel. Ici, CodePen va automatiquement faire la liaison entre mes deux codes et je n'aurai donc pas besoin de préciser `script src="..."` dans mon code. De votre côté, dans votre éditeur, vous devrez évidemment continuer à appeler votre code JavaScript depuis votre page HTML pour que tout fonctionne.

d) La place du code et l'ordre d'exécution de celui-ci

Il y a une autre facteur dont je n'ai pas encore parlé et qu'il faut absolument prendre en compte et comprendre lorsqu'on ajoute du code JavaScript dans nos pages HTML qui est l'ordre d'exécution du code par le navigateur.

Il est possible que ce que je vais expliquer là vous semble complexe et abstrait et c'est tout à fait normal : c'est un peu tôt dans votre apprentissage pour vous expliquer ce mécanisme. Pas d'inquiétude, cependant, nous aurons l'occasion d'en reparler plus tard dans ce cours. Pour le moment, essayez simplement de faire votre maximum pour visualiser ce qu'il se passe.

Ici, il va avant tout être important de bien comprendre que par défaut, un navigateur va lire et exécuter le code dans l'ordre de son écriture.

Plus précisément, lorsque le navigateur arrive à un élément `script`, il va stopper le traitement du reste du HTML jusqu'à ce que le code JavaScript soit chargé dans la page et exécuté.

Nos codes JavaScript ci-dessus ont besoin des éléments `button` de notre page HTML pour fonctionner. En effet, les codes `getElementById('b1')` et `getElementById('b2')` vont récupérer les éléments dont les `id` sont « b1 » et « b2 » pour les manipuler.

Cela risque de poser problème dans le cas présent car si notre code JavaScript est exécuté avant que le code HTML de nos boutons ne soit traité par le navigateur, il ne fonctionnera puisqu'il cherchera à utiliser des éléments qui n'existent pas encore.

C'est la raison pour laquelle, lorsque j'ai choisi d'insérer le code JavaScript directement dans la page HTML au sein d'éléments `script`, j'ai été obligé d'entourer le code JavaScript qui affiche la boîte d'alerte déclaré dans l'élément `head` par le code `document.addEventListener('DOMContentLoaded', function(){});`. Ce code indique en effet au navigateur qu'il doit d'abord charger tout le contenu HTML avant d'exécuter le JavaScript à l'intérieur de celui-ci.

Dans ce même exemple, mon deuxième élément `script` était lui placé en fin de `body` et est donc par défaut exécuté après le reste du code. Il n'y avait donc pas de problème dans ce cas.

Notez que le même problème va avoir lieu dans le cas où on fait appel à un fichier JavaScript externe par défaut : selon l'endroit dans le code où le fichier est demandé, il pourra ne pas fonctionner s'il utilise du code HTML pas encore défini.

Ce souci est la raison pour laquelle il a longtemps été recommandé de placer ses éléments `script` juste avant la balise fermante de l'élément `body`, après tout code HTML.

Cette façon de faire semble en effet résoudre le problème à priori mais n'est pas toujours optimale en termes de performances.

En effet résumons ce qu'il se passe dans ce cas :

1. Le navigateur commence à analyser (ou à traiter) le code HTML ;
2. L'analyseur du navigateur rencontre un élément `script` ;
3. Le contenu JavaScript est demandé et téléchargé (dans le cas où il se situe dans un fichier externe) puis exécuté. Durant tout ce temps, l'analyseur bloque

l'affichage du HTML, ce qui peut dans le cas où le script est long ralentir significativement le temps d'affichage de la page ;

4. Dès que le JavaScript a été exécuté, le contenu HTML finit d'être analysé et est affiché.

Ce problème précis de temps d'attente de chargement des fichiers JavaScript va pouvoir être résolu en grande partie grâce au téléchargement asynchrone des données qui va pouvoir être ordonné en précisant un attribut `async` ou `defer` dans nos éléments `script`.

Le téléchargement asynchrone est une notion complexe et nous l'étudierons donc beaucoup plus tard dans ce cours. Pour le moment, retenez simplement que nous n'allons pouvoir utiliser les attributs `async` et `defer` que dans le cas où on fait appel à des fichiers JavaScript externes (c'est-à-dire à du code JavaScript stocké dans des fichiers séparés).

C'est une raison supplémentaire qui nous fera préférer l'enregistrement du code JavaScript dans des fichiers séparés.

5) Commentaires, indentation et syntaxe de base en JavaScript

a) Les commentaires en JavaScript

Comme pour l'immense majorité des langages de programmation, on va également pouvoir commenter en JavaScript.

Les commentaires sont des lignes de texte (des indications) placées au milieu d'un script et servant à documenter le code, c'est-à-dire à expliquer ce que fait tel ou tel bout de script et éventuellement comment le manipuler.

Ces indications ne seront pas lues par le navigateur et seront donc invisibles pour les visiteurs (sauf s'ils affichent le code source de la page).

Commenter va donc servir aux développeurs à se repérer plus facilement dans un script, à le lire et à le comprendre plus vite. Cela peut être utile à la fois pour vous même si vous travaillez sur des projets complexes ou pour d'autres développeurs si vous êtes amené à distribuer votre code un jour ou l'autre.

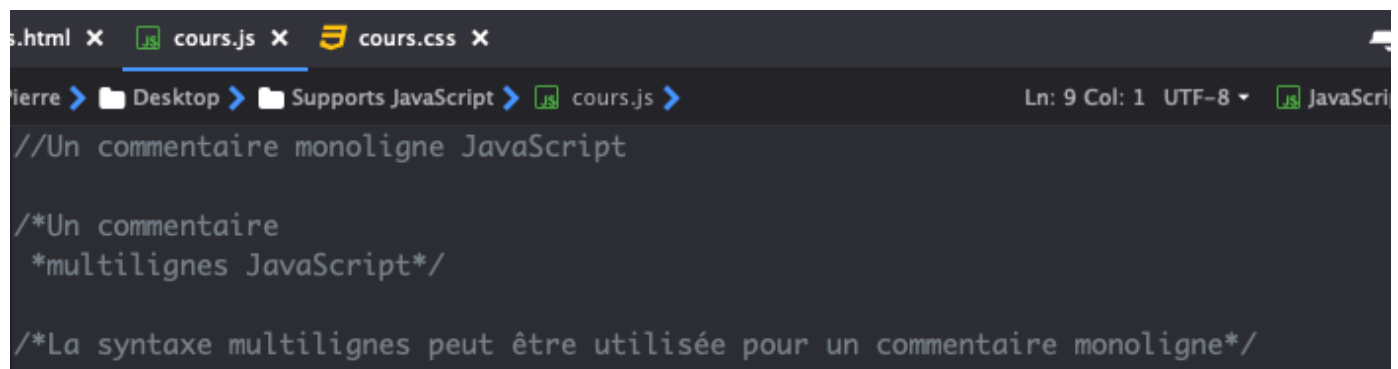
En JavaScript, il existe deux types de commentaires qui vont s'écrire différemment : les commentaires mono-ligne et les commentaires multi-lignes.

Notez que la syntaxe des commentaires multi-lignes peut être utilisée pour écrire un commentaire mono-ligne. Vous pouvez donc vous contenter de n'utiliser que cette syntaxe.

Pour écrire un commentaire multilignes, il faudra entourer le texte de notre commentaire avec la syntaxe suivante `/* */`.

Pour écrire un commentaire monoligne, on utilisera un double slash `//` qui sera suivi du texte de notre commentaire (ou éventuellement la syntaxe multilignes).

Dans l'exemple ci-dessous, on crée trois commentaires dans notre fichier `cours.js` qui utilisent les deux syntaxes et couvrent tous les cas d'utilisation :



```
//Un commentaire monoligne JavaScript

/*Un commentaire
 *multilignes JavaScript*/

/*La syntaxe multilignes peut être utilisée pour un commentaire monoligne*/
```

b) L'indentation en JavaScript

L'indentation correspond au fait de décaler certaines lignes de code par rapport à d'autres. Cela est généralement utilisé pour rendre son code plus lisible et donc plus simple à comprendre.

Pour savoir comment et quand indenter, il suffit de penser en termes de hiérarchie comme on le faisait déjà en HTML.

c) Un premier point sur la syntaxe de base du JavaScript

Avant de véritablement apprendre à coder en JavaScript, j'aimerais discuter d'un point qui divise la communauté des développeurs JavaScript : l'usage du point-virgule.

En effet, sur le net, vous verrez certains tutoriels affirmer que « toute instruction en JavaScript doit être terminée explicitement avec un point-virgule » et d'autres auteurs dire que « les points virgules ne sont souvent pas nécessaires dans le code ».

Avant tout, vous devez savoir qu'un code JavaScript est composé d'instructions. On va avoir différents types d'instruction en JavaScript : la déclaration d'une variable ou d'une fonction, la création d'une boucle, d'une condition, etc. vont toutes être des instructions.

Le point-virgule est généralement utilisé en informatique pour indiquer la fin d'une instruction, c'est-à-dire pour séparer deux instructions l'une de l'autre et cela va également être le cas en JavaScript.

L'idée ici est que le langage JavaScript est très bien fait et ne nous oblige pas strictement à utiliser un point-virgule pour notifier la fin de chaque instruction. En effet, le JavaScript va être capable de « deviner » quand une instruction de termine et va ajouter automatiquement des points-virgules là où ça lui semble pertinent.

C'est la raison pour laquelle certains développeurs se passent tant que possible de ces points-virgules. Cependant, il y a une limite majeure à cela.

Celle-ci est que tout langage informatique repose sur un ensemble de règles. Ainsi, les points-virgules ne sont pas ajoutés automatiquement par le JavaScript au hasard mais selon un ensemble de règles précises.

Pour pouvoir se passer des points-virgules, il faut donc déjà bien connaître le langage et les règles d'ajout automatique des points virgules pour créer un code avec une structure qui va pouvoir être interprétée correctement par la JavaScript.

Sans une connaissance parfaite du comportement du JavaScript et des règles d'ajout, on risque d'avoir des résultats inattendus voire un code non fonctionnel puisqu'il est possible que le JavaScript ajoute des points-virgules là où on ne s'y attend pas.

Pour cette raison, nous ajouterons explicitement des points-virgules à la fin de (presque) toutes les instructions dans ce cours.

II- Présentation des variables JavaScript

Dans cette partie, nous allons découvrir ce que sont les variables en JavaScript et apprendre à les manipuler.

1) Qu'est-ce qu'une variable ?

Une variable est un conteneur servant à stocker des informations de manière temporaire, comme une chaîne de caractères (un texte) ou un nombre par exemple.

Le propre d'une variable est de pouvoir varier, c'est-à-dire de pouvoir stocker différentes valeurs au fil du temps et c'est cette particularité qui les rend si utiles.

Notez bien déjà qu'une variable en soi et la valeur qu'elle va stocker sont deux éléments différents et qui ne sont pas égaux. Encore une fois, une variable n'est qu'un conteneur. Vous pouvez imaginer une variable comme une boîte dans laquelle on va pouvoir placer différentes choses au cours du temps.

Les variables sont l'une des constructions de base du JavaScript et vont être des éléments qu'on va énormément utiliser. Nous allons illustrer leur utilité par la suite.

a) Les règles de déclaration des variables en JavaScript

Une variable est donc un conteneur ou un espace de stockage temporaire qui va pouvoir stocker une valeur. Lorsqu'on stocke une valeur dans une variable, on dit également qu'on assigne une valeur à une variable.

Pour pouvoir utiliser les variables et illustrer leur intérêt, il va déjà falloir les créer. Lorsqu'on crée une variable en JavaScript, on dit également qu'on « déclare » une variable.

Pour déclarer une variable en JavaScript, nous allons devoir utiliser le mot clef `var` ou le mot clef `let` (nous allons expliquer la différence entre les deux dans la suite de cette leçon) suivi du nom qu'on souhaite donner à notre variable.

Concernant le nom de nos variables, nous avons une grande liberté dans le nommage de celles-ci mais il y a quand même quelques règles à respecter :

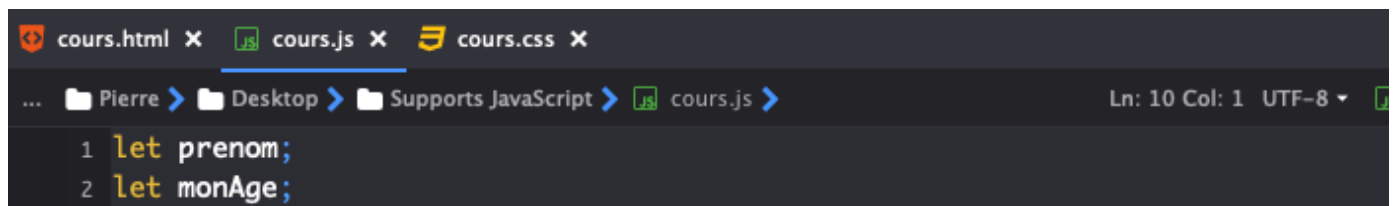
- Le nom d'une variable doit obligatoirement commencer par une lettre ou un underscore (`_`) et ne doit pas commencer par un chiffre ;
- Le nom d'une variable ne doit contenir que des lettres, des chiffres et des underscores mais pas de caractères spéciaux ;
- Le nom d'une variable ne doit pas contenir d'espace.

De plus, notez que le nom des variables est sensible à la casse en JavaScript. Cela signifie que l'usage de majuscules ou de minuscules avec un même nom va permettre de définir des variables différentes. Par exemple, les noms `texte`, `TEXTE` et `tEXTe` vont pouvoir définir des variables différentes.

Enfin, sachez qu'il existe des noms « réservés » en JavaScript. Vous ne pouvez pas utiliser ces noms comme noms pour vos variables, tout simplement car le langage JavaScript les utilise déjà pour désigner différents éléments intégrés au langage. Nous verrons ces différents noms au fil de ce cours.

Vous pouvez également noter qu'on utilise généralement la convention lower camel case pour définir les noms de variable en JavaScript. Cette convention stipule simplement que lorsqu'un nom de variable est composé de plusieurs mots, on colle les mots ensemble en utilisant une majuscule pour chaque mot sauf le premier. Par exemple, si je décide de nommer une variable « monage » j'écrirai en JavaScript `let monAge` ou `var monAge`.

Ci-dessous, on crée nos deux premières variables en utilisant le mot clef `let` dans notre fichier `cours.js` :



```
cours.html x cours.js x cours.css x
... | Pierre | Desktop | Supports JavaScript | cours.js | Ln: 10 Col: 1 UTF-8
1 let prenom;
2 let monAge;
```

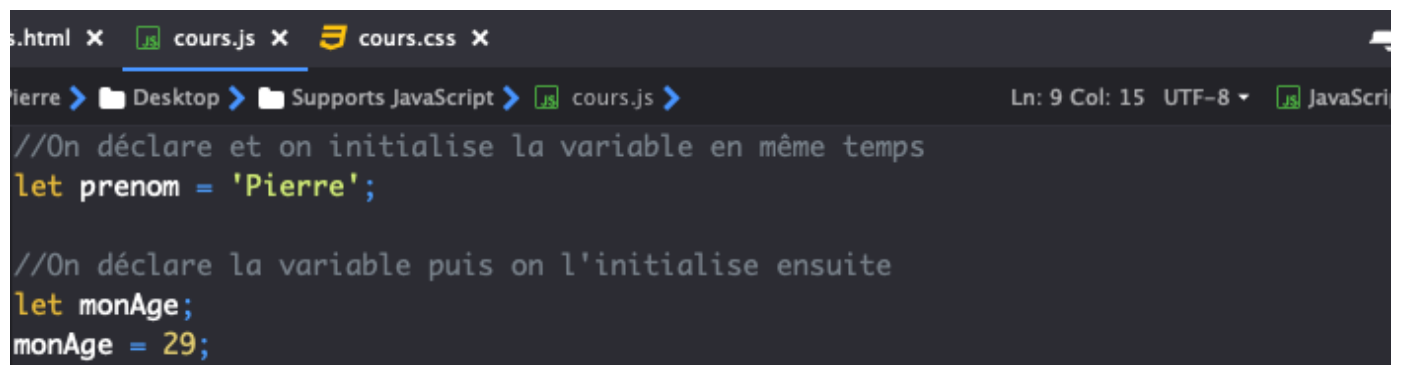
Nos deux premières variables sont désormais créées. Cependant, elles ne stockent aucune valeur pour le moment.

b) Initialiser une variable

Lorsqu'on assigne une valeur pour la première fois à une variable, c'est-à-dire lorsqu'on stocke une valeur dans une variable qui n'en stockait pas encore, on dit également qu'on initialise une variable.

On va pouvoir initialiser une variable après l'avoir déclarée ou au moment de sa déclaration. Les deux façons de faire sont équivalentes en termes de résultat mais il est plus rapide (en termes d'écriture de code) d'initialiser une variable lors de sa déclaration puisque cela nous va nous éviter d'avoir à réécrire le nom de la variable.

Pour initialiser une variable, on utilise l'opérateur `=` qui est dans ce cas non pas un opérateur d'égalité mais un opérateur d'assignation ou d'affectation comme ceci :

A screenshot of a code editor with a dark theme. The top bar shows three tabs: 's.html', 'cours.js', and 'cours.css'. The 'cours.js' tab is active. The breadcrumb navigation shows 'Desktop > Supports JavaScript > cours.js'. The status bar at the bottom right indicates 'Ln: 9 Col: 15 UTF-8 JavaScript'. The code in the editor is as follows:

```
//On déclare et on initialise la variable en même temps
let prenom = 'Pierre';

//On déclare la variable puis on l'initialise ensuite
let monAge;
monAge = 29;
```

Ce point est un point très important à retenir pour éviter les confusions futures et donc je le répète : le signe `=` ne possède pas du tout la même signification que le « égal » mathématique que vous utilisez dans la vie de tous les jours.

Ici, c'est un opérateur d'affectation. Il sert à indiquer qu'on affecte (ou « assigne » ou encore « stocke ») la valeur à droite du signe dans le conteneur à gauche de celui-ci. Encore une fois, la variable n'est pas « égale » à sa valeur.

Vous pouvez également noter deux autres choses intéressantes dans le code ci-dessus : tout d'abord, vous pouvez voir que le mot clef `let` (ou `var`) n'est utilisé et ne doit être utilisé que pour déclarer une variable. Lorsqu'on manipule une variable ensuite, on se contente d'utiliser son nom.

Ensuite, vous remarquez qu'on utilise des apostrophes droites ou guillemets simples pour entourer la valeur « Pierre » mais pas pour la valeur « 29 ». Cela est dû au fait que « Pierre » est une valeur textuelle tandis que « 29 » est un chiffre et ces valeurs ne vont pas pouvoir être

manipulées de la même façon en JavaScript. Nous verrons cela en détail dans la prochaine leçon.

c) Modifier la valeur stockée dans une variable

Le propre d'une variable et l'intérêt principal de celles-ci est de pouvoir stocker différentes valeurs.

Pour affecter une nouvelle valeur dans une variable déjà initialisée, on va se contenter d'utiliser à nouveau l'opérateur d'affectation `=`.

En faisant cela, la nouvelle valeur va venir écraser l'ancienne valeur stockée qui sera alors supprimée.

```
//On déclare et on initialise la variable en même temps
let prenom = 'Pierre';

//On déclare la variable puis on l'initialise ensuite
let monAge;
monAge = 29;

/*On modifie la valeur stockée dans prenom.
 *Notre variable stocke désormais la valeur "Mathilde"*/
prenom = 'Mathilde';
```

Ici, on commence par stocker la valeur « Pierre » dans notre variable `prenom` puis on affecte ensuite la valeur « Mathilde » à notre variable. Cette nouvelle valeur vient écraser l'ancienne car une variable ne peut stocker qu'une valeur à la fois.

d) La différence entre les mots clefs `let` et `var`

Pourquoi possède-t-on deux mots clefs différents pour déclarer des variables en JavaScript ? Cela provient du fait qu'aucun langage n'est parfait ainsi que du fait que les langages informatiques ne sont pas figés mais sont des langages qui évoluent beaucoup et rapidement.

En effet, en informatique, l'augmentation rapide des possibilités (grâce à des connexions plus rapides et à des matériaux de plus en plus performants) pousse les langages à évoluer et notamment à se complexifier et à développer de nouvelles fonctionnalités pour exploiter ces possibilités.

Cette évolution fait que parfois certains langages changent de philosophie de design et modifient certains de leurs composants lorsque ceux-ci deviennent inadaptés.

En effet, en informatique, vous devez absolument comprendre que tout est toujours en mouvement et que ce qui était vrai ou ce qui était considéré comme une bonne pratique il y a 10, 5, 2 ans en arrière ne l'est potentiellement plus aujourd'hui.

Le problème ici est que les différents langages qui ont passé l'épreuve du temps ne peuvent pas du jour au lendemain abandonner complètement certains composants et en définir de nouveaux complètement différents car cela serait, dans le cas d'un langage populaire comme le JavaScript, dramatique pour le web en général.

Effectivement, il faut ici bien comprendre que lorsqu'on crée un site web, on utilise les technologies du moment. Que se passerait-il si certaines fonctionnalités d'un langage étaient brutalement abandonnées et du jour au lendemain plus supportées et donc plus comprises par les navigateurs (dans le cas du JavaScript) qui sont chargées de les exécuter ? La plupart des sites accessibles seraient complètement bogués voire inaccessibles.

Pour cette raison, lorsqu'un langage souhaite faire évoluer ses composants, il doit tenir compte de son héritage et se débrouiller pour faire cohabiter les anciennes fonctionnalités avec les nouvelles au moins le temps que la majorité des propriétaires de sites aient le temps d'implémenter les nouvelles fonctionnalités à la place des anciennes.

Comme vous vous en doutez, dans la plupart des cas, cela prend des années et ce sont généralement dans les faits les navigateurs principaux (Chrome, Firefox, Safari, Explorer) qui « décident » de quand une fonctionnalité est obsolète et qui décident qu'à partir de telle date elle ne sera plus supportée.

Ainsi, la coexistence des mots clefs `var` et `let` en JavaScript est due avant tout à ce souci d'héritage du langage.

Pour être tout à fait précis, lorsque le JavaScript a été créé et jusqu'à il y a quelques années, nous n'avions accès qu'au mot clef `var` qu'on devait utiliser pour déclarer nos variables.

Finalement, les créateurs du JavaScript ont fini par penser que le mot clef `var` pouvait porter à confusion et ont créé un nouveau mot clef pour déclarer les variables : le mot clef `let`.

En même temps qu'un nouveau mot clef a été créé, les créateurs du JavaScript en ont profité pour résoudre quelques problèmes liés à la déclaration de variables en utilisant `var`, ce qui fait que `let` ne va pas nous permettre de créer des variables de la même façon que `var`.

Il existe 3 grandes différences de comportement entre les variables déclarées avec `var` et avec `let` que nous allons illustrer immédiatement.

e) La remontée ou « hoisting » des variables

Lorsqu'on utilise la syntaxe avec `var`, on n'est pas obligé de déclarer la variable avant de la manipuler dans le code, on peut très bien effectuer des manipulations en haut du code et la déclarer en fin de code.

Cela est possible car le JavaScript va traiter les déclarations de variables effectuées avec `var` avant le reste du code JavaScript. Ce comportement est appelé remontée ou hoisting.

Ce comportement a été jugé comme inadapté dans les versions récentes de JavaScript et a donc été corrigé dans la déclaration de variables avec `let` : les variables utilisant la syntaxe `let` doivent obligatoirement être déclarées avant de pouvoir être utilisées.

Le but de ce comportement est de pousser les développeurs à créer des scripts plus compréhensibles et plus clairs en apportant de la structure au code avec notamment la déclaration des différentes variables au début de chaque script.

```
//Ceci fonctionne
prenom = 'Pierre';
var prenom;

//Ceci ne fonctionne pas et renvoie une erreur
nom = 'Giraud';
let nom;
```

f) La redéclaration de variables

Avec l'ancienne syntaxe `var`, on avait le droit de déclarer plusieurs fois une même variable en utilisant à chaque fois `var` (ce qui avait pour effet de modifier la valeur stockée).

La nouvelle syntaxe avec `let` n'autorise pas cela. Pour modifier la valeur stockée dans une variable avec la nouvelle syntaxe, il suffit d'utiliser le nom de la variable et de lui affecter une autre valeur.

Cette décision a été prise une nouvelle fois pour des raisons de clarté et de pertinence du code. En effet, il n'y a aucun intérêt à redéfinir une même variable plusieurs fois.

```
//Ceci fonctionne
var prenom = 'Pierre';
var prenom = 'Mathilde';

//Ceci ne fonctionne pas et renvoie une erreur
let nom = 'Giraud';
let nom = 'Joly';
```

g) La portée des variables

La « portée » d'une variable désigne l'endroit où cette variable va pouvoir être utilisée dans un script. Il est un peu tôt pour vous expliquer ce concept puisque pour bien le comprendre il faut déjà savoir ce qu'est une fonction.

Nous reparlerons donc de cette portée des variables lorsque nous aborderons les fonctions en JavaScript.

Vous pouvez pour le moment retenir si vous le souhaitez que les variables déclarées avec **var** et celles avec **let** au sein d'une fonction ne vont pas avoir la même portée, c'est-à-dire qu'on ne va pas pouvoir les utiliser aux mêmes endroits.

e) Le choix de la déclaration des variables : plutôt avec let ou plutôt avec var

La syntaxe de déclaration des variables avec **let** correspond à la nouvelle syntaxe. La syntaxe avec **var** est l'ancienne syntaxe qui est vouée à disparaître.

Vous devriez donc aujourd'hui toujours utiliser le mot clef **let** pour déclarer vos variables et c'est le mot clef qu'on utilisera dans ce cours.

h) Quelle utilité pour les variables en pratique ?

Les variables vont être à la base de la plupart de nos scripts JavaScript. En effet, il va être très pratique de stocker différents types d'informations dans les variables pour ensuite manipuler simplement ces informations notamment lorsqu'on n'a pas accès à ces informations lorsqu'on crée le script.

Par exemple, on va pouvoir demander à des utilisateurs de nous envoyer des données grâce à la fonction (ou la méthode pour être tout à fait précis mais nous verrons cela plus tard) `prompt()`. Lorsqu'on écrit notre script avec notre fonction `prompt()`, on ne sait pas encore ce que les utilisateurs vont nous envoyer comme données. Dans ce cas, notre script va être créé de manière à ce que les données envoyées soient stockées lors de leur envoi dans des variables qu'on définit. Cela nous permet déjà de pouvoir manipuler les dites variables et par extension les données qu'elles vont stocker.

De même, le fait qu'une même variable puisse stocker plusieurs valeurs dans le temps va être extrêmement utile dans de nombreuses situations. Vous vous souvenez de l'horloge créée au début de ce cours ? Pour créer cette horloge et pour afficher l'heure actuelle, il a fallu utiliser une variable.

Le principe est ici le suivant : je vais chercher l'heure actuelle toutes les secondes et je stocke cette heure dans ma variable que j'affiche ensuite.

A ce propos, il existe de nombreux moyens d'afficher le contenu d'une variable en JavaScript, que ce soit via la console JavaScript du navigateur, en utilisant une fonction `alert()` ou encore en insérant le contenu de notre variable au sein du contenu HTML de notre page. Nous verrons chacune de ces méthodes en détail en temps et en heure, au fil de ce cours.

2) Les types de données en JavaScript

Les variables JavaScript vont pouvoir stocker différents types de valeurs, comme du texte ou un nombre par exemple. Par abus de langage, nous parlerons souvent de « types de variables » JavaScript.

En JavaScript, contrairement à d'autres langages de programmation, nous n'avons pas besoin de préciser à priori le type de valeur qu'une variable va pouvoir stocker. Le JavaScript va en effet automatiquement détecter quel est le type de la valeur stockée dans telle ou telle variable,

et nous allons ensuite pouvoir effectuer différentes opérations selon le type de la variable, ce qui va s'avérer très pratique pour nous !

Une conséquence directe de cela est qu'on va pouvoir stocker différents types de valeurs dans une variable au fil du temps sans se préoccuper d'une quelconque compatibilité. Par exemple, une variable va pouvoir stocker une valeur textuelle à un moment dans un script puis un nombre à un autre moment.

En JavaScript, il existe 7 types de valeurs différents. Chaque valeur qu'on va pouvoir créer et manipuler en JavaScript va obligatoirement appartenir à l'un de ces types. Ces types sont les suivants :

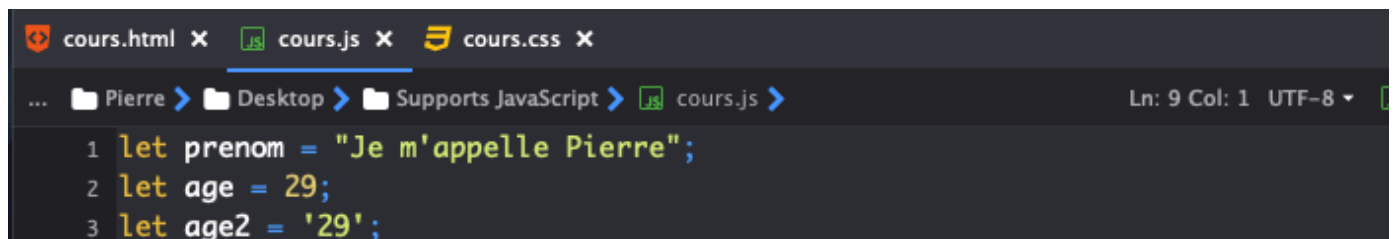
- **String** ou « chaîne de caractères » en français ;
- **Number** ou « nombre » en français ;
- **Boolean** ou « booléen » en français ;
- **Null** ou « nul / vide » en français ;
- **Undefined** ou « indéfini » en français ;
- **Symbol** ou « symbole » en français ;
- **Object** ou « objet » en français ;

Ce que vous devez bien comprendre ici est que les données vont pouvoir être manipulées différemment en fonction de leur type et qu'il est donc essentiel de les connaître pour créer des scripts fonctionnels.

Le type chaîne de caractères ou String

Le premier type de données qu'une variable va pouvoir stocker est le type **String** ou chaîne de caractères. Une chaîne de caractères est une séquence de caractères, ou ce qu'on appelle communément un texte.

Notez que toute valeur stockée dans une variable en utilisant des guillemets ou des apostrophes sera considérée comme une chaîne de caractères, et ceci même dans le cas où nos caractères sont à priori des chiffres comme **"29"** par exemple.

A screenshot of a web browser's developer console or a code editor. The top bar shows three tabs: 'cours.html', 'cours.js', and 'cours.css'. The 'cours.js' tab is active. Below the tabs, a breadcrumb trail shows the file path: '... > Pierre > Desktop > Supports JavaScript > cours.js'. The status bar on the right indicates 'Ln: 9 Col: 1 UTF-8'. The main area displays three lines of JavaScript code:

```
1 let prenom = "Je m'appelle Pierre";  
2 let age = 29;  
3 let age2 = '29';
```

Ici, notre première variable `let prenom` stocke la chaîne de caractère « Je m'appelle Pierre ». Notre deuxième variable `let age`, quant à elle, stocke le nombre 29. En revanche, notre troisième variable `let age2` stocke la chaîne de caractères « 29 » et non pas un nombre.

En effet, l'utilisation de guillemets ou d'apostrophe fait qu'une valeur est immédiatement considérée comme une chaîne de caractères, quelle que soit cette valeur.

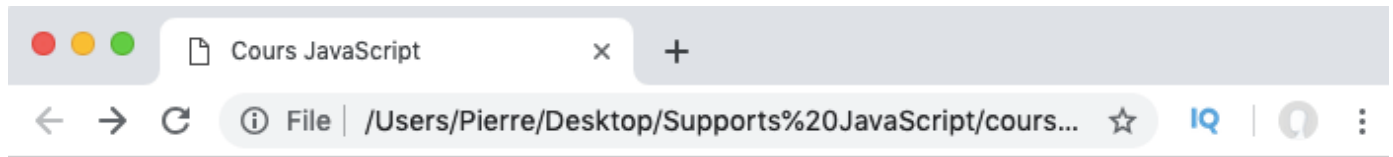
Pour s'en convaincre, on peut utiliser la fonction `typeof` qui nous permet de vérifier le type d'une valeur (éventuellement contenue dans une variable). On va écrire la valeur à tester juste après cet opérateur et celui-ci va renvoyer le type de la valeur.

```
let prenom = "Je m'appelle Pierre";
let age = 29;
let age2 = '29';

document.getElementById('p1').innerHTML = 'Type de prenom : ' + typeof prenom;
document.getElementById('p2').innerHTML = 'Type de age : ' + typeof age;
document.getElementById('p3').innerHTML = 'Type de age2 : ' + typeof age2;
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, user-scalable=no">
    <link rel="stylesheet" href="cours.css">
    <script src='cours.js' async></script>
  </head>

  <body>
    <h1>Titre principal</h1>
    <p>Un paragraphe</p>
    <p id='p1'></p>
    <p id='p2'></p>
    <p id='p3'></p>
  </body>
</html>
```



Titre principal

Un paragraphe

Type de prenom : string

Type de age : number

Type de age2 : string

Encore une fois, n'essayez pas ici de comprendre tout le script car ça n'a pas d'intérêt pour le moment. Ce qui nous intéresse ici sont les résultats renvoyés et comme vous pouvez le constater la variable `let age2` contient bien une valeur considérée comme une chaîne.

Une note pour les plus curieux d'entre vous : vous pouvez retenir que `document.getElementById(id)` nous permet d'accéder à l'élément HTML qui possède l'`id` précisé. Ensuite, `innerHTML` nous permet d'injecter du texte dans l'élément. Dans le cas présent, on injecte « Type de ... » suivi du type de la variable qui est renvoyé par `typeof`.

Un point sur l'utilisation des guillemets et apostrophes droits et sur l'échappement

Dans le code au-dessus, vous pouvez également voir que j'ai utilisé des guillemets droits doubles pour entourer la valeur de la variable `let prenom` et des guillemets droits simples ou apostrophes pour entourer la valeur de `let age2`.

En JavaScript, on va pouvoir utiliser indifféremment des guillemets droits ou des apostrophes pour entourer une chaîne de caractères et ces deux méthodes vont être strictement équivalentes à la différence d'autres langages comme le PHP par exemple.

Faites attention cependant à un point : si votre chaîne contient un caractère qui est le même que le délimiteur de chaîne choisi, il faudra neutraliser ce caractère en l'échappant au moyen d'un antislash ou changer de délimiteur.

Imaginons par exemple que j'utilise des apostrophes pour délimiter la valeur « je m'appelle Pierre » stockée dans `let prenom`. Dans ce cas, le JavaScript va par défaut penser que l'apostrophe dans « m'appelle » correspond à la fin de la chaîne.

Pour lui indiquer que cette apostrophe fait partie de la chaîne et qu'il ne doit pas être considéré comme le délimiteur de fin, on va « l'échapper », c'est-à-dire neutraliser sa signification spéciale qui est ici « délimiteur de chaîne ». Pour cela, il va suffire de faire précéder l'apostrophe en question par un caractère antislash.

Notez que l'antislash est considéré comme le caractère d'échappement dans de nombreux langages informatique. Notez également que si votre chaîne contient des apostrophes incurvées ou des guillemets non droits, il ne sera pas nécessaire de les échapper puisque seuls les apostrophes et guillemets droits sont reconnus comme des délimiteurs de chaîne par le JavaScript.

Regardez plutôt les exemples suivants pour bien comprendre les différents cas possibles :

```
//Délimiteurs non trouvés dans la chaîne = rien à échapper
let a = "Je m'appelle Pierre";

//Délimiteurs non trouvés dans la chaîne (apostrophe non droit) = rien à échapper
let b = 'Je m'appelle Pierre';

//Délimiteurs trouvés dans la chaîne = on échappe le caractère en question
let c = 'Je m\'appelle Pierre';

//Délimiteurs non trouvés dans la chaîne = rien à échapper
let d = "Je m'appelle « Pierre »";

//Délimiteurs trouvés dans la chaîne = on échappe les caractères en question
let e = "Je m'appelle \"Pierre\"";
```

Le type nombre ou Number

Les variables en JavaScript vont également pouvoir stocker des nombres. En JavaScript, et contrairement à la majorité des langages, il n'existe qu'un type prédéfini qui va regrouper tous les nombres qu'ils soient positifs, négatifs, entiers ou décimaux (à virgule) et qui est le type `Number`.

Pour affecter une valeur de type Number à une variable, on n'utilise ni guillemet ni apostrophe.

```
let x = 10; //x stocke un entier positif
let y = -2; //y stocke un entier négatif
let z = 3.14; //z stocke un nombre décimal positif
```

Attention ici : lorsque nous codons nous utilisons toujours des notations anglo-saxonnes, ce qui signifie qu'il faut remplacer nos virgules par des points pour les nombres décimaux.

Le type de données booléen (Boolean)

Une variable en JavaScript peut encore stocker une valeur de type **Boolean**, c'est-à-dire un booléen.

Un booléen, en algèbre, est une valeur binaire (soit 0, soit 1). En informatique, le type booléen est un type qui ne contient que deux valeurs : les valeurs **true** (vrai) et **false** (faux).

Ce type de valeur peut sembler plus compliqué à appréhender à première vue. Pas d'inquiétude, nous allons souvent l'utiliser par la suite car il va nous être particulièrement utile en valeur de test pour vérifier si le test est validé ou non.

Une nouvelle fois, faites bien attention : pour qu'une variable stocke bien un booléen, il faut lui faire stocker la valeur **true** ou **false**, sans guillemets ou apostrophes car dans le cas contraire le JavaScript considèrera que c'est la chaîne de caractères « true » ou « false » qui est stockée et on ne pourra plus effectuer les mêmes opérations avec ces valeurs.

```
let vrai = true; //Stocke le booléen true
let faux = false; //Stocke le booléen false

/*On demande au JavaScript d'évaluer la comparaison "8 > 4". Comme 8 est bien
*strictement supérieur à 4, le JavaScript renvoie true en résultat. On
*stocke ensuite ce résultat (le booléen true) dans la variable let resultat*/
let resultat = 8 > 4;
```

Le troisième exemple est un peu plus complexe à comprendre. Ici, vous devez comprendre que l'affectation se fait en dernier et que la comparaison est faite avant. Lorsqu'on écrit « 8 > 4 », (qui signifie 8 strictement supérieur à 4) on demande au JavaScript d'évaluer cette comparaison.

Si la comparaison est vérifiée (si elle est vraie), alors JavaScript renvoie le booléen `true`. Dans le cas contraire, il renvoie le booléen `false`. On stocke ensuite le booléen renvoyé dans la variable `let resultat`.

Nous aurons largement le temps de nous familiariser avec ce type de construction dans la suite de ce cours et notamment lors de l'étude des boucles et des conditions.

Les types de valeurs Null et Undefined

Les types de valeurs `Null` et `Undefined` sont des types un peu particuliers car ils ne contiennent qu'une valeur chacun : les valeurs `null` et `undefined`.

La valeur `null` correspond à l'absence de valeur ou du moins à l'absence de valeur connue. Pour qu'une variable contienne `null`, il va falloir stocker cette valeur qui représente donc l'absence de valeur de manière explicite.

La valeur `null` va être utile dans certains cas où on souhaite explicitement indiquer une absence de valeur connue. Il va cependant falloir qu'on ait un peu plus d'expérience avec le JavaScript pour montrer de manière pratique l'intérêt de cette valeur.

La valeur `undefined` correspond à une variable « non définie », c'est-à-dire une variable à laquelle on n'a pas affecté de valeur.

Cette définition peut vous paraître similaire à celle de `null` et pourtant ces deux valeurs ont une signification différente.

Si on déclare une variable sans lui attribuer de valeur, alors son type sera `Undefined`. Si on déclare une variable et qu'on lui passe `null`, alors son type sera `Object`.


```

<!DOCTYPE html>
<html>
  <head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, user-scalable=no">
    <link rel="stylesheet" href="cours.css">
    <script src='cours.js' async></script>
  </head>

  <body>
    <h1>Titre principal</h1>
    <p>Un paragraphe</p>
    <p id='p1'></p>
    <p id='p2'></p>
    <p id='p3'></p>
  </body>
</html>

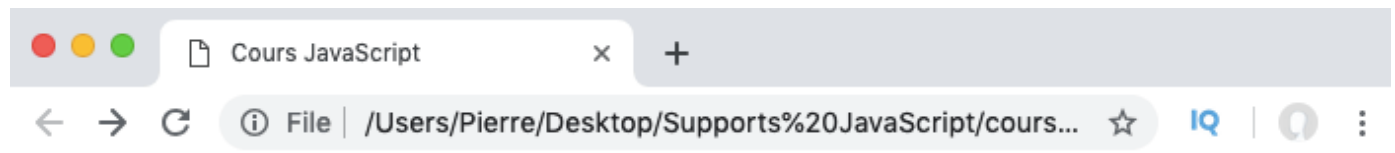
```

```

let nul = null;
let ind;

document.getElementById('p1').innerHTML = 'Type de nul : ' + typeof nul;
document.getElementById('p2').innerHTML = 'Type de ind : ' + typeof ind;

```



Titre principal

Un paragraphe

Type de nul : object

Type de ind : undefined

Le type renvoyé par `typeof` pour `null` est ici `Object` (objet en français). Les objets sont des types complexes de valeurs que nous étudierons plus tard dans ce cours.

Pour le moment, vous pouvez simplement retenir que `null` ne devrait pas être de type `Object` mais bien de type `Null` et que cela est considéré comme une erreur du langage

JavaScript par la majorité des développeurs. Cependant, historiquement, ça a toujours été le cas.

Les types de valeurs Object (objet) et Symbol (symbole)

Les objets sont des structures complexes qui vont pouvoir stocker plusieurs valeurs en même temps et que nous étudierons plus tard dans ce cours car il s'agit là d'un sujet relativement complexe à appréhender pour un débutant.

Notez qu'en JavaScript, contrairement à d'autres langages, les tableaux sont avant tout des objets et sont des valeurs de type **Object**.

Finalement, nous parlerons du type de données **Symbol** après avoir vu les objets car ces deux types sont liés.

3) Présentation des opérateurs arithmétiques et d'affectation JavaScript

Qu'est-ce qu'un opérateur ?

Un opérateur est un symbole qui va être utilisé pour effectuer certaines actions notamment sur les variables et leurs valeurs.

Par exemple, l'opérateur ***** va nous permettre de multiplier les valeurs de deux variables, tandis que l'opérateur **=** va nous permettre d'affecter une valeur à une variable.

Il existe différents types d'opérateurs qui vont nous servir à réaliser des opérations de types différents. Les plus fréquemment utilisés sont :

- Les opérateurs arithmétiques ;
- Les opérateurs d'affectation / d'assignation ;
- Les opérateurs de comparaison ;
- Les opérateurs d'incrément et de décrémentation ;
- Les opérateurs logiques ;
- L'opérateur de concaténation ;

- L'opérateur ternaire ;
- l'opérateur virgule.

Pour le moment, nous allons nous concentrer particulièrement sur les opérateurs arithmétiques et d'affectation ou d'assignation. Nous étudierons les autres par la suite lorsqu'on devra les utiliser dans certaines structures JavaScript.

Les opérateurs arithmétiques

Les opérateurs arithmétiques vont nous permettre d'effectuer toutes sortes d'opérations mathématiques entre les valeurs de type **Number** contenues dans différentes variables.

Le fait de pouvoir réaliser des opérations entre variables va être très utile dans de nombreuses situations puisqu'en JavaScript nous allons souvent utiliser des nombres pour traiter des données ou calculer de nouvelles valeurs.

Nous allons pouvoir utiliser les opérateurs arithmétiques suivants en JavaScript :

Opérateur	Nom de l'opération associée
+	Addition
—	Soustraction
*	Multiplication
/	Division
%	Modulo (reste d'une division euclidienne)
**	Exponentielle (élévation à la puissance d'un nombre par un autre)

Avant d'utiliser les opérateurs arithmétiques, clarifions ce que sont le modulo et l'exponentielle.

Le modulo correspond au reste entier d'une division euclidienne. Par exemple, lorsqu'on divise 5 par 3, le résultat est 1 et il reste 2 dans le cas d'une division euclidienne. Le reste, 2, correspond justement au modulo.

L'exponentielle correspond à l'élévation à la puissance d'un nombre par un autre nombre. La puissance d'un nombre est le résultat d'une multiplication répétée de ce nombre par lui-même. Par exemple, lorsqu'on souhaite calculer 2 à la puissance de 3 (qu'on appelle également « 2 exposant 3 »), on cherche en fait le résultat de 2 multiplié 3 fois par lui-même c'est-à-dire $2*2*2 = 8$.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, user-scalable=no">
    <link rel="stylesheet" href="cours.css">
    <script src='cours.js' async></script>
  </head>

  <body>
    <h1>Titre principal</h1>
    <p>Un paragraphe</p>
    <p id='p1'></p>
    <p id='p2'></p>
    <p id='p3'></p>
  </body>
</html>
```

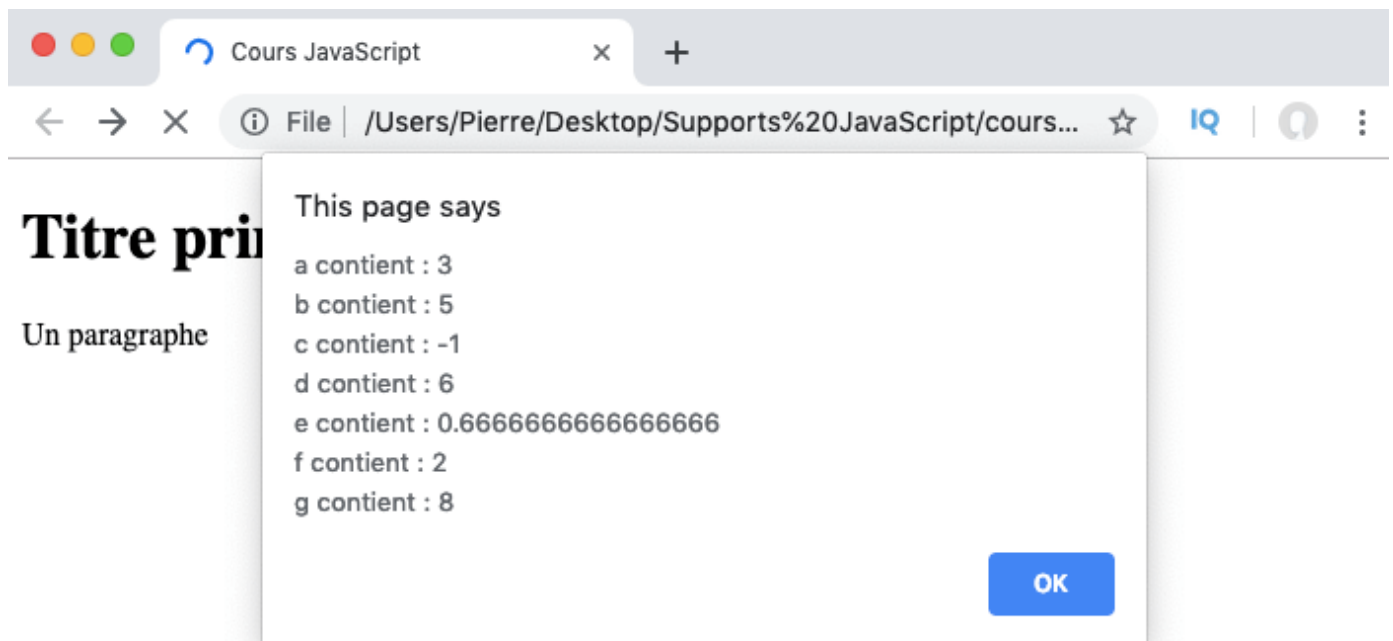
```

let x = 2;
let y = 3;
let z = 4;

let a = x + 1; //a stocke 2 + 1 = 3
let b = x + y; //b stocke 2 + 3 = 5
let c = x - y; //c stocke 2 - 3 = -1
let d = x * y; //d stocke 2 * 3 = 6
let e = x / y; //e stocke 2 / 3
let f = 5 % 3; //f stocke le reste de la division euclidienne de 5 par 3
let g = x ** 3; //g stocke 2^3 = 2 * 2 * 2 = 8

/*On affiche les résultats dans une boîte d'alerte en utilisant l'opérateur
 *de concaténation "+". On retourne à la ligne dans l'affichage avec "\n"*/
alert('a contient : ' + a +
      '\nb contient : ' + b +
      '\nc contient : ' + c +
      '\nd contient : ' + d +
      '\ne contient : ' + e +
      '\nf contient : ' + f +
      '\ng contient : ' + g);

```



Ici, on effectue quelques opérations mathématiques de base et on affiche les résultats dans une boîte d'alerte. Vous pouvez déjà noter qu'on utilise à nouveau le signe `+` pour afficher les résultats mais cette fois-ci cet opérateur est utilisé en tant qu'opérateur de concaténation. Nous reparlerons de cela dans la suite de cette leçon. Notez également l'utilisation du `\n` qui sert en JavaScript à retourner à la ligne à la manière d'un élément HTML `br`.

Priorité des calculs et utilisation des parenthèses

Concernant les règles de calcul, c'est-à-dire l'ordre de priorité des opérations, celui-ci va être le même qu'en mathématiques : l'élévation à la puissance va être prioritaire sur les autres opérations, tandis que la multiplication, la division et le modulo vont avoir le même ordre de priorité et être prioritaires sur l'addition et la soustraction qui ont également le même niveau de priorité.

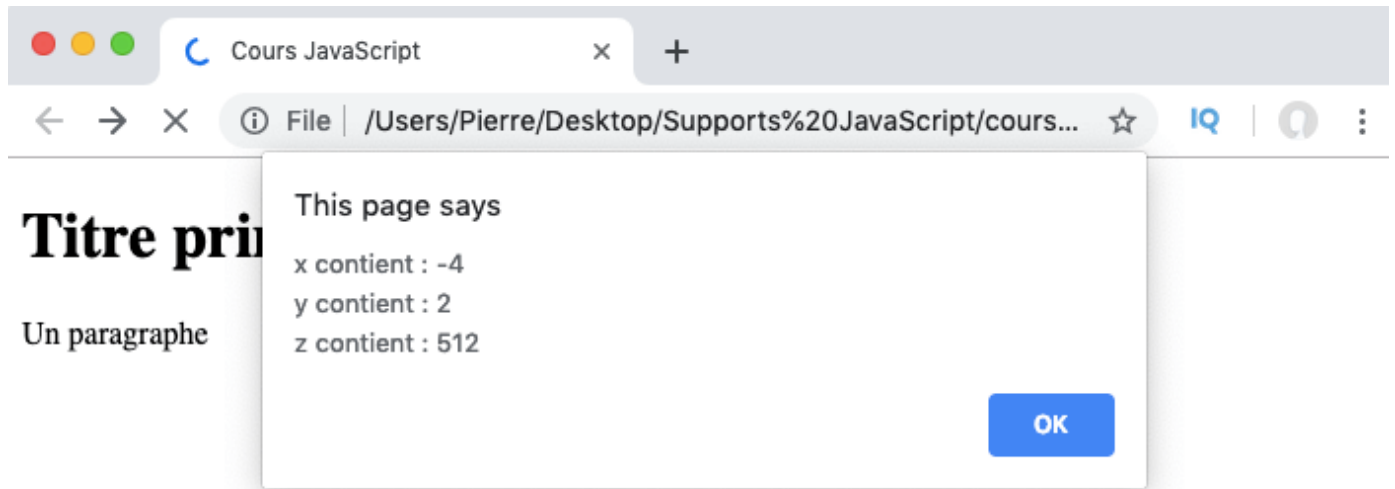
Si deux opérateurs ont le même ordre de priorité, alors c'est leur sens d'association qui va décider du résultat. Pour les opérateurs arithmétiques, le sens d'association correspond à l'ordre de leur écriture à l'exception de l'élévation à la puissance qui sera calculée en partant de la fin.

Ainsi, si on écrit `let x = 1 - 2 - 3`, la variable `let x` va stocker la valeur -4 (les opérations se font de gauche à droite). En revanche, si on écrit `let z = 2 ** 3 ** 2`, la variable `let z` stockera 512 qui correspond à 2 puissance 9 puisqu'on va commencer par calculer $3 ** 2 = 9$ dans ce cas.

Nous allons finalement, comme en mathématiques, pouvoir forcer l'ordre de priorité en utilisant des couples de parenthèses pour indiquer qu'une opération doit se faire avant toutes les autres :

```
let x = 1 - 2 - 3; //Calcule (1 - 2) - 3 = -1 - 3 = - 4
let y = 1 - (2 - 3); //Calcule 1 - (2 - 3) = 1 - (- 1) = 1 + 1 = 2
let z = 2 ** 3 ** 2; //Calcule 3 ** 2 = 3 * 3 = 9 puis 2 ** 9 = 512

/*On affiche les résultats dans une boîte d'alerte en utilisant l'opérateur
 *de concaténation "+". On retourne à la ligne dans l'affichage avec "\n"*/
alert('x contient : ' + x +
      '\ny contient : ' + y +
      '\nz contient : ' + z);
```



Les opérateurs d'affectation

Les opérateurs d'affectation vont nous permettre, comme leur nom l'indique, d'affecter une certaine valeur à une variable.

Nous connaissons déjà bien l'opérateur d'affectation le plus utilisé qui est le signe `=`. Cependant, vous devez également savoir qu'il existe également des opérateurs « combinés » qui vont effectuer une opération d'un certain type (comme une opération arithmétique par exemple) et affecter en même temps.

Vous pourrez trouver ci-dessous quelques-uns de ces opérateurs qui vont être le plus utiles pour nous pour le moment :

Opérateur	Définition
<code>+=</code>	Additionne puis affecte le résultat
<code>-=</code>	Soustrait puis affecte le résultat
<code>*=</code>	Multiplie puis affecte le résultat

Opérateur	Définition
/=	Divise puis affecte le résultat
%=	Calcule le modulo puis affecte le résultat

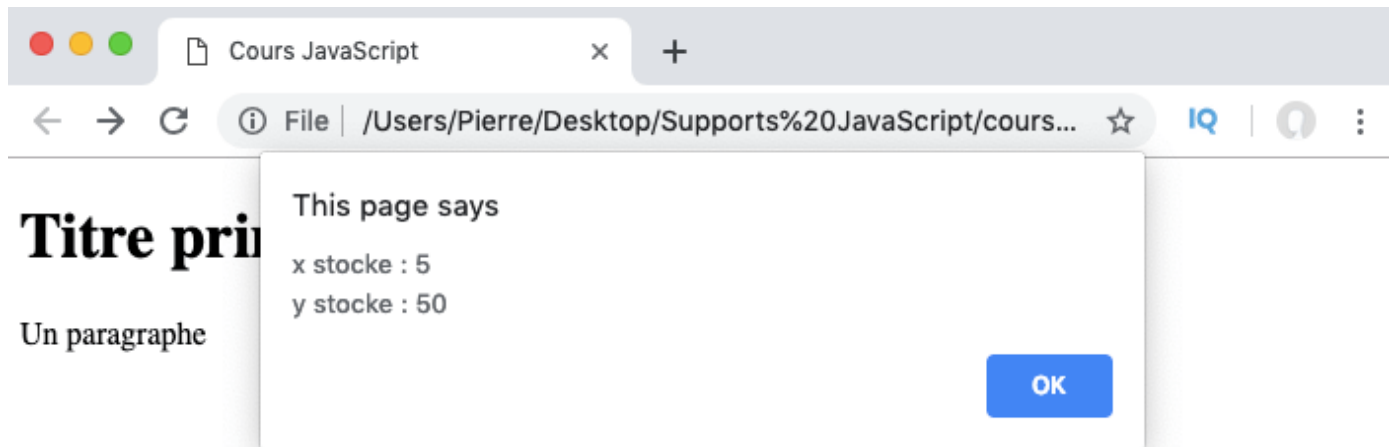
Illustrons immédiatement cela et voyons comment se servir de ces opérateurs :

```
let x = 2; //x stocke 2
let y = 10; //y stocke 10

/*On ajoute 3 à la valeur stockée précédemment par x (2) puis on
 *affecte le résultat à x. x stocke désormais 2 + 3 = 5*/
x += 3;

/*On multiplie la valeur de y (10) par celle de z (5) puis on affecte
 *le résultat à y. y stocke désormais 10 * 5 = 50*/
y *= x;

alert('x stocke : ' + x + '\ny stocke : ' + y);
```



Ici, vous devez bien comprendre que les opérateurs d'affectation combinés nous permettent d'effectuer deux types d'opérations à la suite. Dans l'exemple ci-dessus, on réalise des opérations arithmétiques puis d'affectation.

Ainsi, l'opérateur `+=` par exemple va nous permettre d'ajouter une valeur à la valeur contenue dans une variable puis d'affecter le résultat dans cette même variable. La nouvelle valeur va ainsi écraser la valeur précédente.

4) La concaténation et les littéraux de gabarits en JavaScript

Dans cette nouvelle leçon, nous allons expliquer en détail ce qu'est la concaténation et comment fonctionne l'opérateur de concaténation qu'on a déjà pu voir dans les leçons précédentes.

Nous allons également voir une autre façon d'entourer nos données en JavaScript avec les littéraux de gabarits (appelés en anglais « template literals » ou « template strings »).

La concaténation en JavaScript

Concaténer signifie littéralement « mettre bout à bout ». La concaténation est un mot généralement utilisé pour désigner le fait de rassembler deux chaînes de caractères pour en former une nouvelle.

En JavaScript, l'opérateur de concaténation est le signe `+`. Faites bien attention ici : lorsque le signe `+` est utilisé avec deux nombres, il sert à les additionner. Lorsqu'il est utilisé avec autre chose que deux nombres, il sert d'opérateur de concaténation.

La concaténation va nous permettre de mettre bout-à-bout une chaîne de caractères et la valeur d'une variable par exemple. Sans opérateur de concaténation, on ne peut pas en effet utiliser une chaîne de caractères et une variable côte à côte car le JavaScript ne saurait pas reconnaître les différents éléments.

Notez une chose intéressante relative à la concaténation en JavaScript ici : si on utilise l'opérateur `+` pour concaténer une chaîne de caractères puis un nombre, alors le JavaScript va considérer le nombre comme une chaîne de caractères.

Regardez les exemples suivants pour bien comprendre :


```

<!DOCTYPE html>
<html>
  <head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, user-scalable=no">
    <link rel="stylesheet" href="cours.css">
    <script src='cours.js' async></script>
  </head>

  <body>
    <h1>Titre principal</h1>
    <p>Un paragraphe</p>
    <p id='p1'></p>
    <p id='p2'></p>
    <p id='p3'></p>
  </body>
</html>

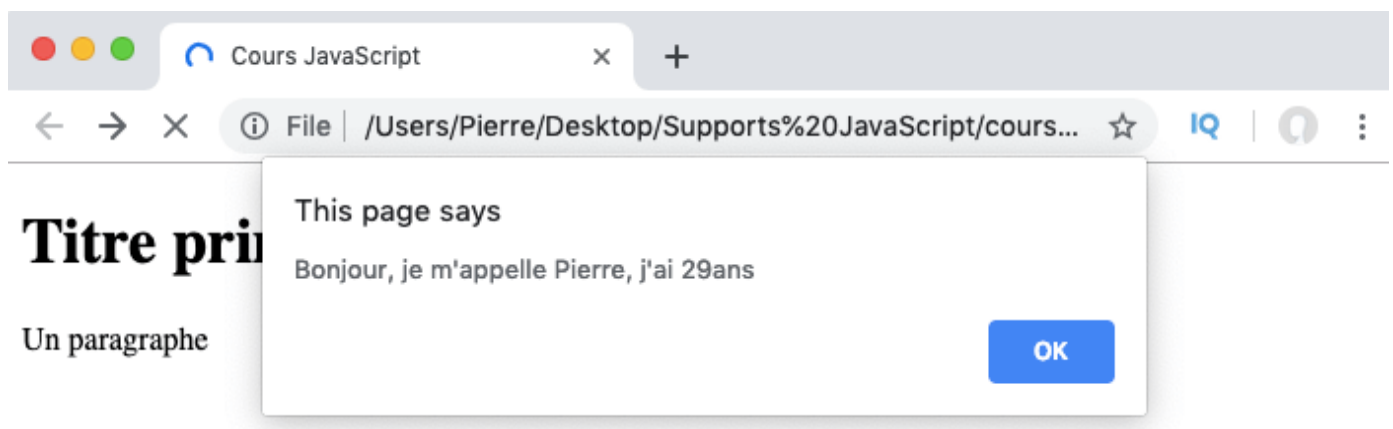
```

```

let x = 28 + 1; //Le signe "+" est ici un opérateur arithmétique
let y = 'Bonjour';
let z = x + 'ans'; //Le signe "+" est ici un opérateur de concaténation

alert(y + ', je m\'appelle Pierre, j\'ai ' + z);

```



Ici, le `+` utilisé dans la valeur de `let x` est un opérateur arithmétique qui sert à additionner car les deux opérandes (les deux éléments situés à gauche et à droite de l'opérateur) sont des nombres.

Dans `let z`, en revanche, le signe `+` est utilisé pour concaténer, c'est-à-dire pour mettre bout à bout la valeur de `let x`, c'est-à-dire 29 et le mot « ans ». La variable `let z` stocke ainsi « 29 ans ».

Finalement, on utilise des signes `+` au sein de notre instruction `alert` pour pouvoir afficher côté à côté le contenu de nos variables et du texte.

Pour être tout à fait précis, vous pouvez retenir que lorsqu'on utilise le signe `+`, le JavaScript va considérer tout ce qui se situe après une chaîne de caractères comme des chaînes de caractères. Ainsi, si on écrit `'un' + 2 + 4`, le JavaScript concaténera en `'un24'` tandis que si on écrit `2 + 4 + 'un'`, la valeur finale sera `'6un'`.

Les littéraux de gabarits

On a vu plus tôt dans ce cours qu'il fallait en JavaScript toujours entourer nos chaînes de caractères (nos textes) avec des apostrophes ou des guillemets droits.

Il existe en fait une troisième manière introduite récemment d'entourer des chaînes de caractères en JavaScript qui va utiliser des accents graves ```.

La grande différence entre l'utilisation d'accent graves ou l'utilisation d'apostrophes ou de guillemets est que toute expression placée entre les accents graves va être interprétée en JavaScript. Pour le dire simplement : tout ce qui renvoie une valeur va être remplacé par sa valeur.

Cela signifie notamment qu'on va pouvoir placer du texte et des variables ensemble sans avoir besoin d'utiliser d'opérateur de concaténation puisque les variables vont être interprétées, c'est-à-dire remplacées par leur valeur.

Pour que cela fonctionne bien, il va cependant falloir respecter une certaine syntaxe : il va falloir placer les expressions entre `${}` et `}`.

Prenons immédiatement un exemple pour bien comprendre. Pour cela, je vais créer deux variables et ensuite utiliser une boîte d'alerte pour afficher leur valeur et la somme des valeurs avec un texte. On va faire ça de deux façons différentes : en utilisant la concaténation d'abord puis en utilisant les littéraux de gabarits.

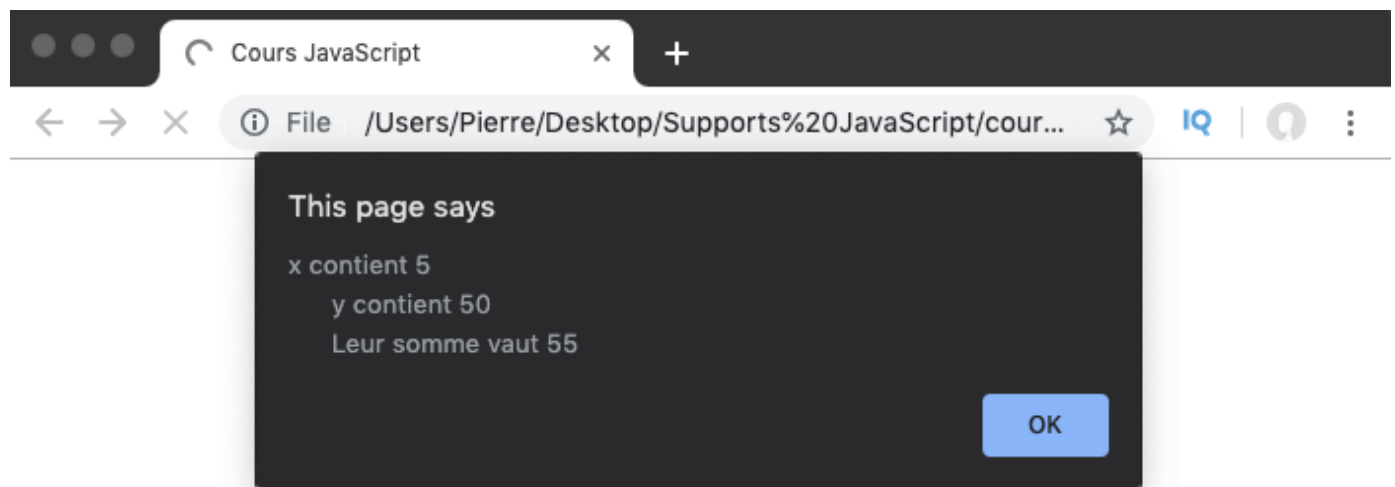
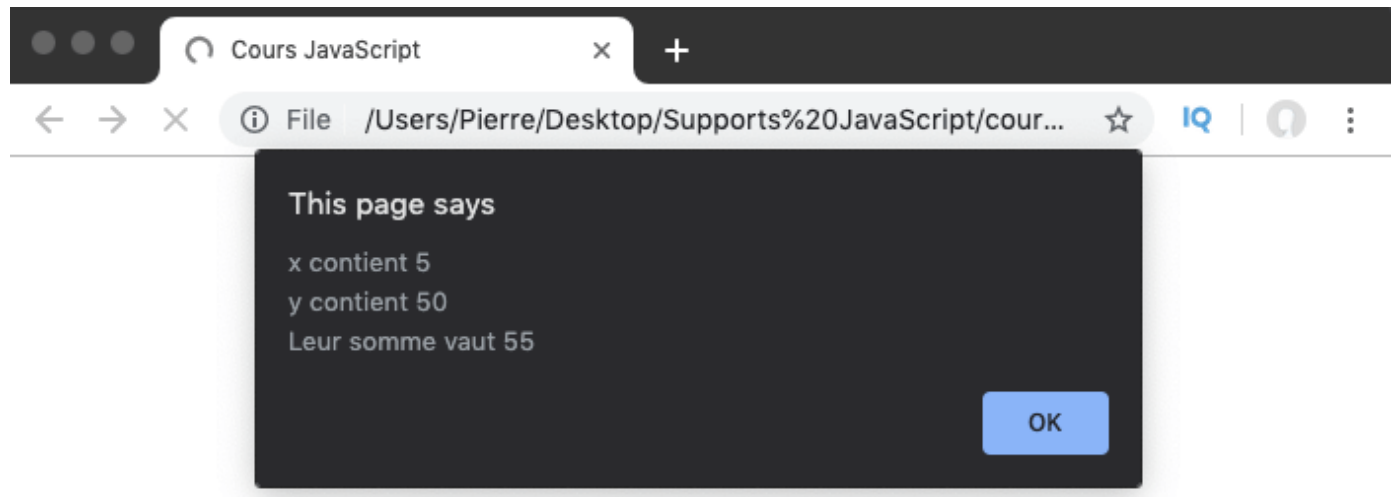
```

let x = 5;
let y = 50;

alert('x contient ' + x +
      '\ny contient ' + y +
      '\nLeur somme vaut ' + (x + y));

alert(`x contient ${x}
      y contient ${y}
      Leur somme vaut ${x + y}`);

```



Comme vous pouvez le remarquer, les deux instructions `alert()` renvoient un résultat équivalent. Notez également que l'utilisation des littéraux de gabarits conserve les retours à la ligne et les décalages dans le résultat final.

En effet, dans le premier `alert()` utilisant la concaténation, vous pouvez voir qu'on a utilisé des `\n` qui servent à matérialiser un retour à la ligne en JavaScript.

Cela n'est pas nécessaire lorsqu'on utilise les littéraux de gabarits : les retours à la ligne et décalages sont conservés. C'est la raison pour laquelle les deuxième et troisième ligne de l'instruction `alert()` sont décalées dans ce cas (l'indentation du code est conservée).

Les littéraux de gabarits vont donc pouvoir s'avérer utiles lorsqu'on doit manipuler des chaînes et des variables voire des fonctions. Pour des raisons de clarté, cependant, je ne les utiliserai que très peu dans la suite de ce cours (leur usage est généralement conseillé pour des développeurs relativement avancés).

5) Les constantes en JavaScript

Le JavaScript supporte depuis quelques années l'utilisation des constantes. Nous allons voir ce que sont ces éléments de langage dans cette leçon.

Définition et utilité des constantes en JavaScript

Une constante est similaire à une variable au sens où c'est également un conteneur pour une valeur. Cependant, à la différence des variables, on ne va pas pouvoir modifier la valeur d'une constante.

En effet, une fois qu'une valeur est attribuée à une constante, celle-ci est attribuée de façon définitive et ne va pas pouvoir être modifiée. C'est d'ailleurs de là que les constantes portent leur nom : car leur valeur est constante.

Les constantes vont être très utiles dans le cadre d'un script qui va réutiliser souvent la même valeur mais qui doit toujours utiliser cette valeur exactement. Dans ce cas-là, plutôt que de réécrire la valeur à chaque fois, nous allons stocker la valeur dans une constante et utiliser la constante.

Dans ce cas-là, utiliser une constante va rendre notre script plus clair car on pourra rapidement identifier la valeur utilisée et également plus facilement maintenable car dans le cas où l'on doit modifier le script et cette valeur en particulier un jour, on n'aura alors qu'à modifier la constante plutôt que de modifier toutes les occurrences de la valeur dans le script.

Déclarer une constante en JavaScript

Pour créer ou déclarer une constante en JavaScript, nous allons utiliser le mot clef `const`.

On va pouvoir déclarer une constante exactement de la même façon qu'une variable à la différence qu'on va utiliser `const` à la place de `let`.

Notez qu'il faut obligatoirement initialiser une constante lors de sa déclaration, c'est-à-dire lui passer une valeur immédiatement faute de quoi une erreur sera retournée.

```
const prenom = 'Pierre';  
const age = 29;
```