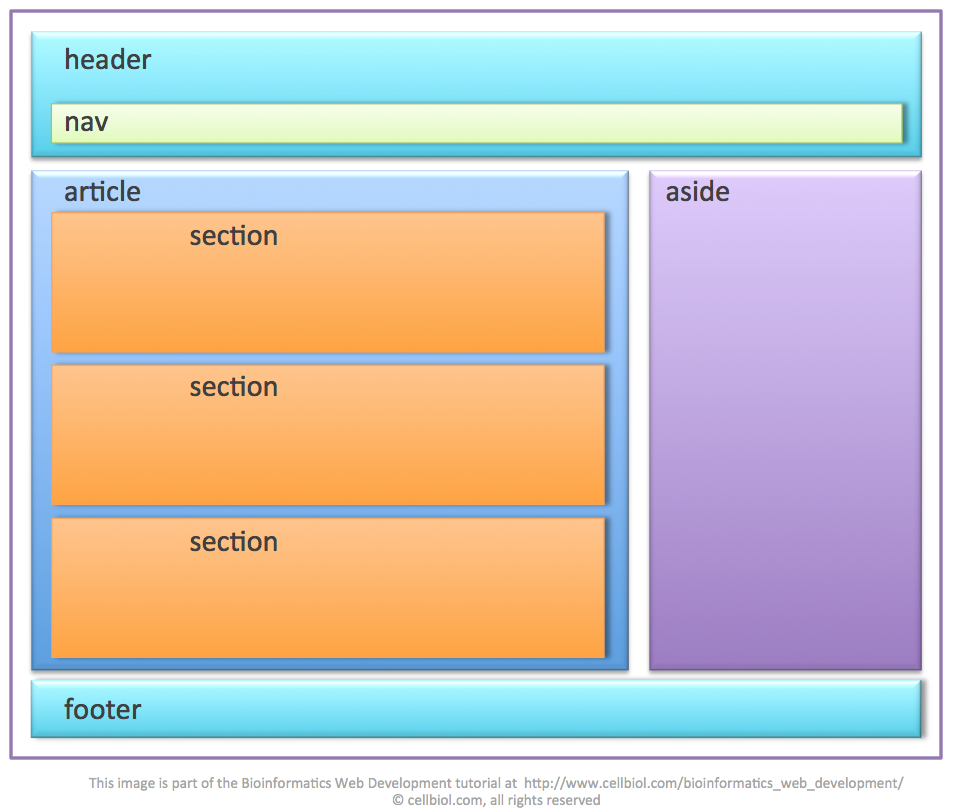
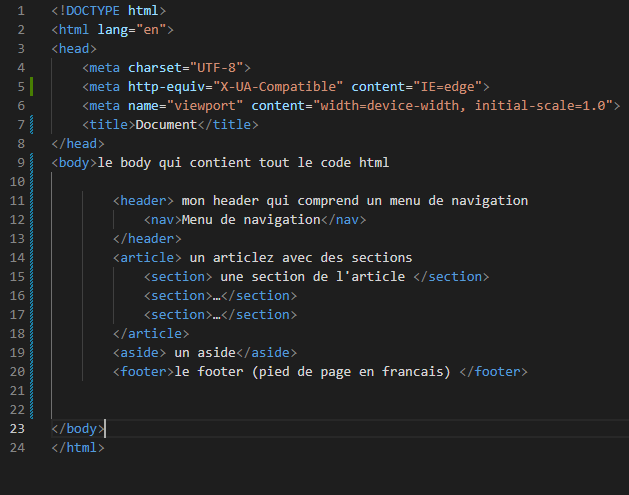
Javascript ECF v2

Votre ECF "Front-end" se déroulera le mercredi 06/07/2022 ce sera l'occasion de faire le point sur la maitrise du HTML5 et CSS3 pour faire des composants d'interface comme un menu animé à partir de liste puce, ou un menu par Onglet par exemple ou encore un formulaire bien mise en forme, un diaporama. La partie JS portera sur les contrôles de saisies, manipulation du DOM, et la consommation d'API JSON (méthode JS Fetch() ou Objet XHR) pour alimenter un diaporama ou une liste d'annonces ou de produits( générés dynamiquement). Durée de l’épreuve de 07h00. ( 4 heures le matin, 3 heures l'après-midi)

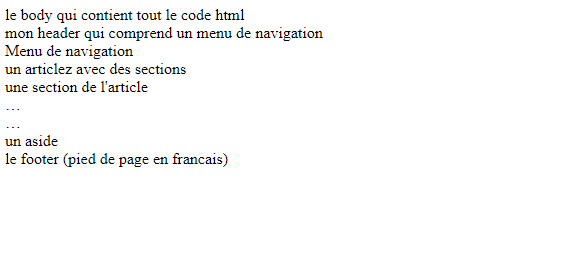
Plan type d’une page html5 :



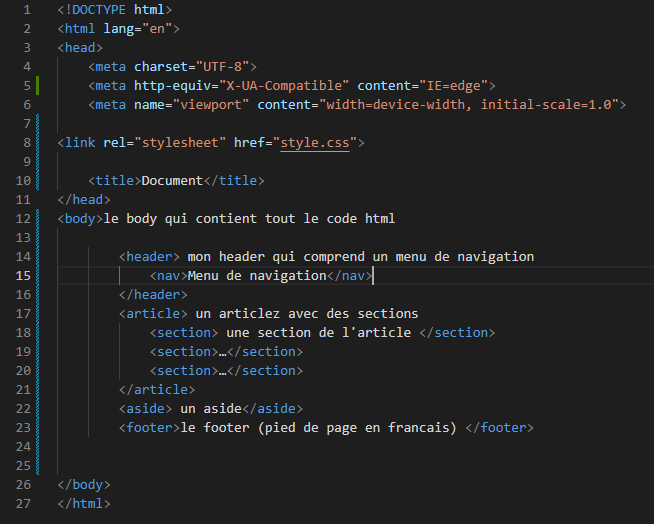
Voici un plan de page html5 qu’on pourrait écrire en code HTML5 comme décrit ci-dessous :



Regardez le résultat cela est moche non ?



En effet aucun style CSS n’est appliqué. Pour l’appliqué il faut relier un fichier CSS au html Reprenons cette exemple : comme vous pouvez le voir j’ai rajoutez une balise link qui sert à relier un fichier css du même nom ici (style.css).



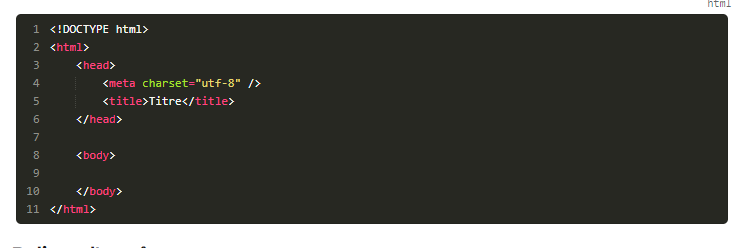
**Mémento HTML5**

**Balises de premier niveau**

Les balises de premier niveau sont les principales balises qui structurent une page HTML. Elles sont indispensables pour réaliser le « code minimal » d'une page web.

| **Balise** | **Description** |
| --- | --- |
| <html> | Balise principale |
| <head> | En-tête de la page |
| <body> | Corps de la page |

Code minimal d'une page HTML :



**Balises d'en-tête**

Ces balises sont toutes situées dans l'en-tête de la page web, c'est-à-dire entre <head>  et </head>  :

| **Balise** | **Description** |
| --- | --- |
| <link /> | Liaison avec une feuille de style |
| <meta /> | Métadonnées de la page web (charset, mots-clés, etc.) |
| <script> | Code JavaScript |
| <style> | Code CSS |
| <title> | Titre de la page |

**Balises de structuration du texte**

| **Balise** | **Description** |
| --- | --- |
| <abbr> | Abréviation |
| <blockquote> | Citation (longue) |
| <cite> | Citation du titre d'une œuvre ou d'un évènement |
| <q> | Citation (courte) |
| <sup> | Exposant |
| <sub> | Indice |
| <strong> | Mise en valeur forte |
| <em> | Mise en valeur normale |
| <mark> | Mise en valeur visuelle |
| <h1> | Titre de niveau 1 |
| <h2> | Titre de niveau 2 |
| <h3> | Titre de niveau 3 |
| <h4> | Titre de niveau 4 |
| <h5> | Titre de niveau 5 |
| <h6> | Titre de niveau 6 |
| <img /> | Image |
| <figure> | Figure (image, code, etc.) |
| <figcaption> | Description de la figure |
| <audio> | Son |
| <video> | Vidéo |
| <source> | Format source pour les balises <audio>  et <video> |
| <a> | Lien hypertexte |
| <br /> | Retour à la ligne |
| <p> | Paragraphe |
| <hr /> | Ligne de séparation horizontale |
| <address> | Adresse de contact |
| <del> | Texte supprimé |
| <ins> | Texte inséré |
| <dfn> | Définition |
| <kbd> | Saisie clavier |
| <pre> | Affichage formaté (pour les codes sources) |
| <progress> | Barre de progression |
| <time> | Date ou heure |

**Balises de listes**

Cette section énumère toutes les balises HTML permettant de créer des listes (listes à puces, listes numérotées, listes de définitions…)

| **Balise** | **Description** |
| --- | --- |
| <ul> | Liste à puces, non numérotée |
| <ol> | Liste numérotée |
| <li> | Élément de la liste à puces |
| <dl> | Liste de définitions |
| <dt> | Terme à définir |
| <dd> | Définition du terme |

**Balises de tableau**

| **Balise** | **Description** |
| --- | --- |
| <table> | Tableau |
| <caption> | Titre du tableau |
| <tr> | Ligne de tableau |
| <th> | Cellule d'en-tête |
| <td> | Cellule |
| <thead> | Section de l'en-tête du tableau |
| <tbody> | Section du corps du tableau |
| <tfoot> | Section du pied du tableau |

**Balises de formulaire**

| **Balise** | **Description** |
| --- | --- |
| <form> | Formulaire |
| <fieldset> | Groupe de champs |
| <legend> | Titre d'un groupe de champs |
| <label> | Libellé d'un champ |
| <input /> | Champ de formulaire (texte, mot de passe, case à cocher, bouton, etc.) |
| <textarea> | Zone de saisie multiligne |
| <select> | Liste déroulante |
| <option> | Élément d'une liste déroulante |
| <optgroup> | Groupe d'éléments d'une liste déroulante |

**Balises sectionnantes**

Ces balises permettent de construire le squelette de notre site web.

| **Balise** | **Description** |
| --- | --- |
| <header> | En-tête |
| <nav> | Liens principaux de navigation |
| <footer> | Pied de page |
| <section> | Section de page |
| <article> | Article (contenu autonome) |
| <aside> | Informations complémentaires |

**Balises génériques**

Les balises génériques sont des balises qui n'ont pas de sens sémantique.

En effet, toutes les autres balises HTML ont un *sens* : <p>  signifie « paragraphe », <h2>  signifie « sous-titre », etc.  
Parfois, on a besoin d'utiliser des balises génériques (aussi appelées ***balises universelles***) car aucune des autres balises ne convient. On utilise le plus souvent des balises génériques pour construire son design.

Il y a deux balises génériques : l'une est inline, l'autre est block.

| **Balise** | **Description** |
| --- | --- |
| <span> | Balise générique de type inline |
| <div> | Balise générique de type block |

Ces balises ont un intérêt uniquement si vous leur associez un attribut class  , id  ou style  :

* **class**  : indique le nom de la classe CSS à utiliser.
* **id**  : donne un nom à la balise. Ce nom doit être unique sur toute la page car il permet d'identifier la balise. Vous pouvez vous servir de l'ID pour de nombreuses choses, par exemple pour créer un lien vers une ancre, pour un style CSS de type ID, pour des manipulations en JavaScript, etc.
* **style**  : cet attribut vous permet d'indiquer directement le code CSS à appliquer. Vous n'êtes donc pas obligé d'avoir une feuille de style à part, vous pouvez mettre directement les attributs CSS. Notez qu'il est préférable de ne pas utiliser cet attribut et de passer à la place par une feuille de style externe, car cela rend votre site plus facile à mettre à jour par la suite.

Ces trois attributs ne sont pas réservés aux balises génériques : vous pouvez aussi les utiliser sans aucun problème dans la plupart des autres balises.

**Memento css**

Je résume ici la plupart des propriétés de **mise en forme du texte**.

Qu'est-ce que la mise en forme de texte ? C'est tout ce qui touche à la présentation du texte proprement dit : le gras, l'italique, le souligné, la police, l'alignement, etc.

| **Propriété** | **Valeurs (exemples)** | **Description** |
| --- | --- | --- |
| font-family | *police1, police2, police3*, serif, sans-serif, monospace | Nom de police |
| @font-face | *Nom et source de la police* | Police personnalisée |
| font-size | 1.3em, 16px, 120%... | Taille du texte |
| font-weight | bold, normal | Gras |
| font-style | italic, oblique, normal | Italique |
| text-decoration | underline, overline, line-through, blink, none | Soulignement, ligne au-dessus, barré ou clignotant |
| font-variant | small-caps, normal | Petites capitales |
| text-transform | capitalize, lowercase, uppercase | Capitales |
| font | - | Super propriété de police. Combine : font-weight  , font-style  , font-size  , font-variant  , font-family  . |
| text-align | left, center, right, justify | Alignement horizontal |
| vertical-align | baseline, middle, sub, super, top, bottom | Alignement vertical (cellules de tableau ou éléments inline-block  uniquement) |
| line-height | 18px, 120%, normal... | Hauteur de ligne |
| text-indent | 25px | Alinéa |
| white-space | pre, nowrap, normal | Césure |
| word-wrap | break-word, normal | Césure forcée |
| text-shadow | 5px 5px 2px blue *(horizontale, verticale, fondu, couleur)* | Ombre de texte |

**Propriétés de couleur et de fond**

| **Propriété** | **Valeurs (exemples)** | **Description** |
| --- | --- | --- |
| color | *nom*, rgb(rouge,vert,bleu), rgba(rouge,vert,bleu,transparence), #CF1A20... | Couleur du texte |
| background-color | *Identique à color* | Couleur de fond |
| background-image | url('image.png') | Image de fond |
| background-attachment | fixed, scroll | Fond fixe |
| background-repeat | repeat-x, repeat-y, no-repeat, repeat | Répétition du fond |
| background-position | *(x y)*, top, center, bottom, left, right | Position du fond |
| background | - | Super propriété du fond. Combine :  background-image  , background-repeat  , background-attachment  , background-position |
| opacity | 0.5 | Transparence |

**Propriétés des boîtes**

| **Propriété** | **Valeurs (exemples)** | **Description** |
| --- | --- | --- |
| width | 150px, 80%... | Largeur |
| height | 150px, 80%... | Hauteur |
| min-width | 150px, 80%... | Largeur minimale |
| max-width | 150px, 80%... | Largeur maximale |
| min-height | 150px, 80%... | Hauteur minimale |
| max-height | 150px, 80%... | Hauteur maximale |
| margin-top | 23px | Marge en haut |
| margin-left | 23px | Marge à gauche |
| margin-right | 23px | Marge à droite |
| margin-bottom | 23px | Marge en bas |
| margin | 23px 5px 23px 5px *(haut, droite, bas, gauche)* | Super-propriété de marge. Combine : margin-top  , margin-right  , margin-bottom  , margin-left  . |
| padding-left | 23px | Marge intérieure à gauche |
| padding-right | 23px | Marge intérieure à droite |
| padding-bottom | 23px | Marge intérieure en bas |
| padding-top | 23px | Marge intérieure en haut |
| padding | 23px 5px 23px 5px *(haut, droite, bas, gauche)* | Super-propriété de marge intérieure. Combine : padding-top  , padding-right  , padding-bottom  , padding-left  . |
| border-width | 3px | Épaisseur de bordure |
| border-color | *nom*, rgb(rouge,vert,bleu), rgba(rouge,vert,bleu,transparence), #CF1A20... | Couleur de bordure |
| border-style | solid, dotted, dashed, double, groove, ridge, inset, outset | Type de bordure |
| border | 3px solid black | Super-propriété de bordure. Combine ,border-width  , border-color  , border-style  . Existe aussi en versionborder-top  , border-right  , border-bottom  , border-left  . |
| border-radius | 5px | Bordure arrondie |
| box-shadow | 6px 6px 0px black *(horizontale, verticale, fondu, couleur)* | Ombre de boîte |

**Propriétés de positionnement et d'affichage**

| **Propriété** | **Valeurs (exemples)** | **Description** |
| --- | --- | --- |
| display | block, inline, inline-block, table, table-cell, none... | Type d'élément ( block  , inline  , inline-block  , none  …) |
| visibility | visible, hidden | Visibilité |
| clip | rect (0px, 60px, 30px, 0px) *rect (haut, droite, bas, gauche)* | Affichage d'une partie de l'élément |
| overflow | auto, scroll, visible, hidden | Comportement en cas de dépassement |
| float | left, right, none | Flottant |
| clear | left, right, both, none | Arrêt d'un flottant |
| position | relative, absolute, static | Positionnement |
| top | 20px | Position par rapport au haut |
| bottom | 20px | Position par rapport au bas |
| left | 20px | Position par rapport à la gauche |
| right | 20px | Position par rapport à la droite |
| z-index | 10 | Ordre d'affichage en cas de superposition. La plus grande valeur est affichée par-dessus les autres. |

**Propriétés des listes**

| **Propriété** | **Valeurs (exemples)** | **Description** |
| --- | --- | --- |
| list-style-type | disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha, none | Type de liste |
| list-style-position | inside, outside | Position en retrait |
| list-style-image | url('puce.png') | Puce personnalisée |
| list-style | - | Super-propriété de liste. Combine list-style-type  , ,list-style-position  , list-style-image  . |

**Propriétés des tableaux**

| **Propriété** | **Valeurs (exemples)** | **Description** |
| --- | --- | --- |
| border-collapse | collapse, separate | Fusion des bordures |
| empty-cells | hide, show | Affichage des cellules vides |
| caption-side | bottom, top | Position du titre du tableau |

**Autres propriétés**

| **Propriété** | **Valeurs (exemple)** | **Description** |
| --- | --- | --- |
| cursor | crosshair, default, help, move, pointer, progress, text, wait, e-resize, ne-resize, auto... | Curseur de souris |

**Memo javascript**

Une variable permet d’associer un nom à une valeur. La valeur peut prendre plusieurs formes (texte, nombre, boolean, etc.). On la déclare avec le mot clé  let .

*Exemple*:



**Constante**

Une constante est une variable qui a la particularité de ne pas pouvoir changer de valeur dans le temps. On la déclare avec le mot clé  const  .

*Exemple*:

C:\Users\Stagiaire\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Const.png

**Type**

Un type correspond à la nature des valeurs que peut prendre une donnée (string, number, array, etc).

*Exemple*:



Selon son type on peut appliquer divers actions sur une variable :

* Les variables  number  pourront s’additionner, multiplier, etc.
* Les variables  string  pourront être converti en majuscule avec la fonction  maString.toUpperCase()
* Les autres types ont chacun des fonctionnalités associées.

**Programmation orientée Objet (POO)**

La programmation orientée objet est un paradigme (concept) de programmation qui consiste à définir et faire interagir des éléments qu’on appelle “objets”. Un “objet” est un concept, une représentation, une idée qui se rattache au monde physique. Par exemple, un livre, une page de livre, une lettre, etc.

**Classe**

En POO, une classe regroupe les méthodes et propriétés (attributs) qui se rapportent à un objet. Elle définit ce qu’est un objet et ce que l’on peut faire avec.

*Exemple*:*C:\Users\Stagiaire\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Class.png*

**Instance de classe**

L’instance de classe est la création d’un objet unique basée sur la définition d’une classe. Une instance est donc un élément unique créé à partir d’une classe. Par exemple, à partir d’une classe Maison, on peut créer 2 instances qui seront 2 maisons différentes mais basées sur les mêmes plans de construction.

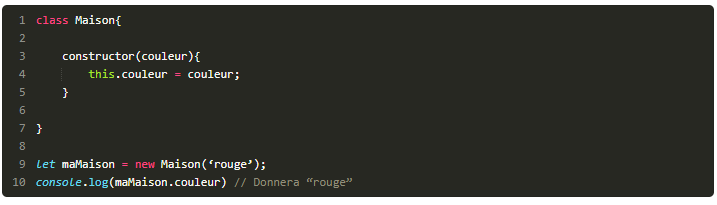
*Exemple*:



**Propriété/attribut de classe**

La propriété (dite aussi *attribut*) de classe est une variable interne à une classe qui pourra évoluer. Par exemple, un objet maison a un nombre de portes de base, mais cette valeur peut évoluer selon les actions appliquées sur cette maison (des travaux, par exemple).

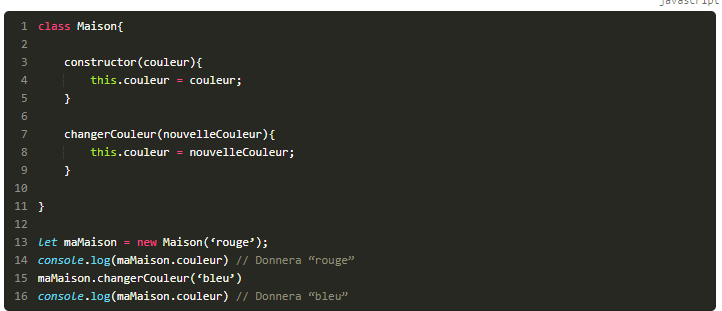
*Exemple*:



**Méthode de classe**

La méthode de classe est une fonction interne à la classe qui permet d'exécuter des actions au sein de la classe instanciée. Par exemple, la méthode “peindre(couleur)” peut changer la propriété “couleur” de l’objet maison.

*Exemple*:



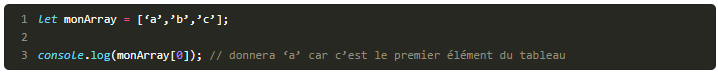
**Collection**

Les collections sont des types de données qui permettent de ranger un ensemble de données dans une seule variable (comme ranger de la donnée dans un tableau). On trouve principalement deux types de collection : les Array et les Object.

**Collection de type Array**

Une collection de type Array permet de créer une liste ordonnée d’éléments. On accède à l’un de ces éléments par son index. Il faut savoir que le premier index est 0 et non pas 1.

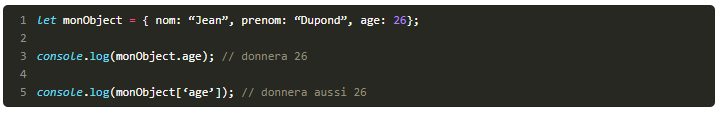
*Exemple*:

**

**Collection de type Object**

Une collection de type Object est une liste non ordonnée qui ne fonctionne pas à partir d’index mais à partir de clé à laquelle est rattachée une valeur. Donc pour récupérer une valeur, on doit renseigner la clé de cette valeur.

*Exemple*:



**Bloc de code**

Un bloc de code est une partie du code commençant par un crochet ouvrant  {  et terminant par un crochet fermant}.

*Exemple*:

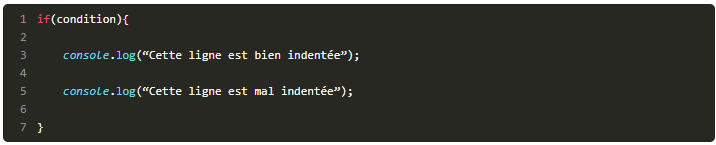


**Indentation**

C’est la mise en forme des lignes de code pour rendre la lecture plus simple et explicite.

La règle est de mettre le code en retrait par rapport à la ligne précédente pour signifier que la ligne d’un niveau inférieur. Dans le cas de l’écriture d’un bloc de code, cela permet de mieux comprendre où commence et où s'arrête chaque bloc.

*Exemple*:

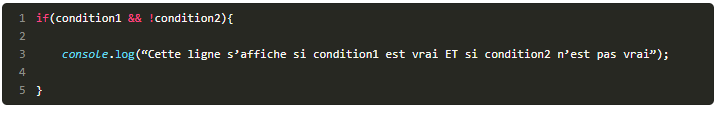


**Opérateurs logiques**

Ce sont des caractères spéciaux qui permettent de faire des conditions plus complexes en liant plusieurs conditions, ou en faisant des comparaisons de valeurs. On retrouve les caractères suivants : &&, ||, <, >, >=, <=, ==, ===, != et !.

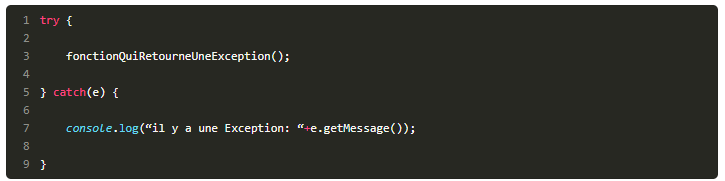
*Exemple*:

**Exception**



Dès qu’il y a une erreur liée à un défaut d'exécution, une erreur est soulevée en JavaScript, on appelle cela une Exception. On le remarque souvent dans la console en rouge. Une Exception contient des informations permettant de comprendre les raisons du problème, et on peut facilement récupérer cette Exception au sein du code pour faire un traitement personnalisé grâce au bloc  try    { }     catch(e){  }  .

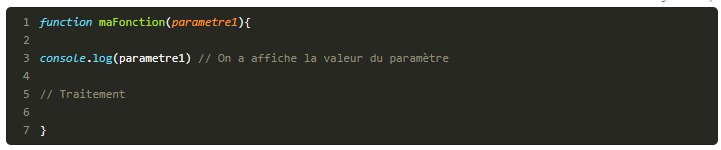
*Exemple*:



**Paramètre de fonction**

Un paramètre est une valeur qu’une fonction attend en entrée. On lui donne un nom comme une variable qui servira au sein de la fonction pour le traitement. Il peut bien sûr y avoir plusieurs paramètres dans une fonction.

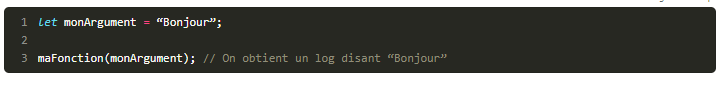
*Exemple*:



**Argument de fonction**

L’argument d’une fonction est la valeur qu’on va injecter en entrée d’une fonction au moment de l'exécution. Ce concept est lié directement à la notion de paramètre de fonction, car en général pour chaque paramètre d’une fonction, on devra apporter un argument.

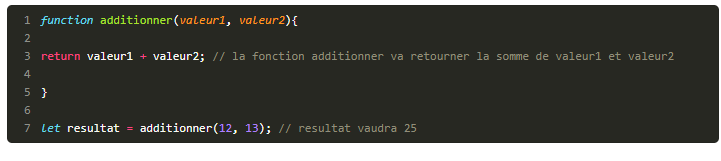
*Exemple*:



**Valeur de retour**

Lors de l'exécution d’une fonction, il y a plusieurs traitements réalisés en son sein, et une valeur finale pourra être retournée avec le mot clé “return”. Cela permet de récupérer la valeur retournée par la fonction dans une variable, par exemple, et de l’exploiter dans le reste du code.

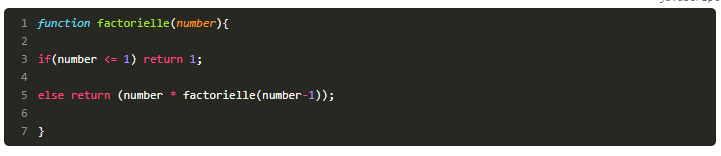
*Exemple*:



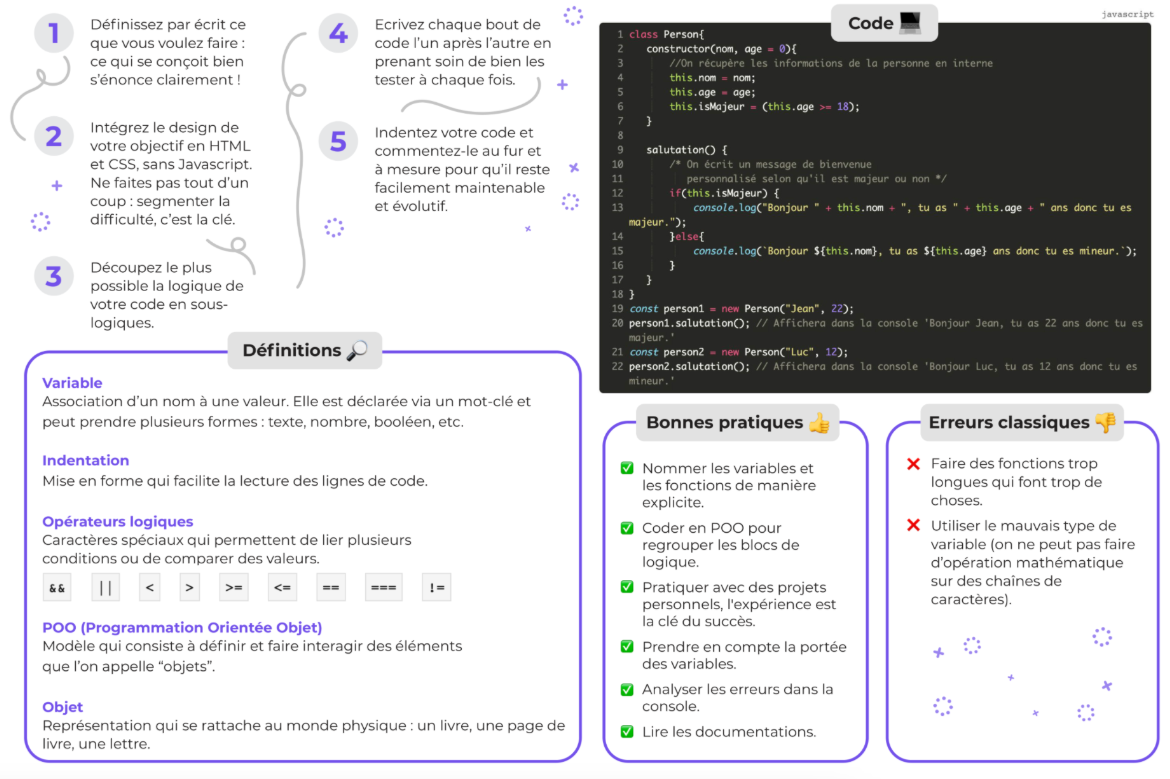
**Récursivité**

La récursivité est un concept dans lequel un élément fait appel à lui-même. Par exemple, une fonction récursive est une fonction qui s'appelle elle-même d'une façon ou d'une autre.

*Exemple*:



***Résumé:***



**Manipulation du DOM en javascript:**

Au-delà de jouer avec des boucles et des chaînes de caractères, JavaScript permet de manipuler un document et de jouer avec le HTML et/ou son CSS, le tout en temps réel.

Le document que nous allons pouvoir manipuler, appelé le DOM *(Document Object Model)* est, par exemple, la page sur laquelle vous vous trouvez à l'instant même ! Il ne s'agit pas de modifier un ensemble de pages, mais bien la page en cours.  
Évidemment, dans le cadre de cet article, nous ne modifierons pas cette page. Cependant, nous supposerons que vous avez une page en cours de développement, sur laquelle nous pourrons appliquer quelques essais.

**DOM**

C'est une arborescence créée par votre navigateur contenant la structure HTML de la page Web chargée, ce qui permet à JavaScript de manipuler ce document.



Le code afficher ici c’est le **DOM**

**querySelector() & querySelectorAll()**

Grâce aux méthodes querySelector() et querySelectorAll(), nous allons pouvoir sélectionner un ou plusieurs éléments du DOM et y appliquer par la suite nos différents ajouts ou modifications.

Elles attendent un argument afin de sélectionner l'élément voulu. Vous pouvez leur passer le nom du tag HTML *(a, section, img, div…)*, le nom d'un sélecteur CSS *(.ma\_classe, .container…)* ou l'ID de l'élément désiré *(#mon\_id, #first, ...)*.

**querySelector()**

Cette méthode sélectionnera le premier élément trouvé selon l'argument passé.

*let* paragraph = *document*.querySelector('p');

Dans cet exemple, le premier élément HTML p repéré sera sélectionné et les suivants seront ignorés.

**querySelectorAll()**

Celle-ci sélectionnera tous les éléments repérés selon l'argument passé. Il faudra utiliser une boucle afin de retrouver chaque élément un à un.

*let* buttons = *document*.querySelectorAll('.btn .btn-danger');

**utilisation du getElementsByTagName(), getElementsByClassName() & getElementById()**

**getElementsByTagName()**

Placer purement le nom du tag HTML que vous souhaitez récupérer. Cette méthode vous retournera une collection HTML, soit un tableau. À lire avec une boucle si vous voulez intervenir sur les différents éléments retrouvés.

*let* paragraphs = *document*.getElementsByTagName('p');

**getElementsByClassName()**

Utilisez la classe CSS afin de retrouver les éléments HTML qui l'utiliseraient. Cette dernière vous retournera une collection HTML, soit un tableau. À lire avec une boucle si vous souhaitez intervenir sur les différents éléments retrouvés.

*let* containers = *document*.getElementsByClassName('container');

**getElementById()**

Utilisez l'ID dans le but de retrouver l'élément HTML qui l'utiliserait. Cette dernière vous retournera un seul élément, le premier trouvé. Pas besoin de boucle.

*let* list = *document*.getElementById('listUser');

**Les propriétés**

Une fois votre ou vos éléments retrouvés, vous avez une multitude de propriétés permettant d'y apporter des modifications. Nous allons en voir quelques-unes, les plus utilisées :

**innerHTML**  
Reprends ou change la syntaxe HTML à l'intérieur de l'élément.

**outerHTML**  
Reprends ou modifie la syntaxe HTML complète de l'élément.

**innertText**  
Reprends ou modifie la syntaxe textuelle à l'intérieur de l'élément.

**textContent**  
Reprends ou modifie la syntaxe textuelle à l'intérieur de l'élément ainsi que des ascendants.

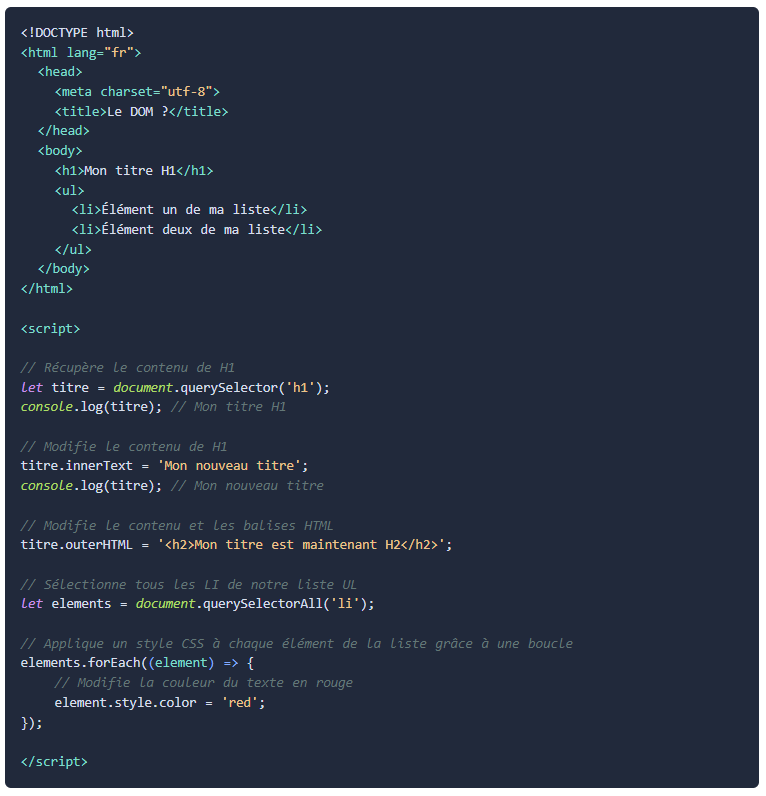
**dataset**  
Reprends, ajoute ou modifie une valeur contenue dans un attribut "data" d'un élément.

**setAttribute/getAttribute**  
Reprends ou modifie n'importe quel attribut d'un élément.

**classList**  
Permet de "jouer" sur les classes CSS d'un élément.

**style**  
Aide à jouer sur le style CSS d'un élément.

lExemple :



**Créer un nouveau nœud**

Jouer avec le DOM pour en modifier des valeurs, ajouter des classes CSS… c'est bien, mais on peut aussi créer de nouveaux nœuds et cela facilement !

Avant de créer un nouveau nœud, nous allons retrouver un élément dans lequel nous mettrons justement cette nouvelle création. Pour l'exemple, j'ajoute une nouvelle li à ma liste.

Dans un premier temps, je reprends ma ul :

*let* liste = *document*.querySelector('ul');

Maintenant, je crée mon nouvel élément :

*let* li = *document*.createElement('li');

J'ajoute le texte que je souhaite dans ce nouvel élément :

li.innerText = 'Element trois de ma liste';

Enfin, je peux l'ajouter à la fin de ma liste :

liste.appendChild(li);

**Supprimer un nœud**

Cette action est assez simple. À partir du moment où vous avez repris un élément, la suppression est facile :

*let* liste = *document*.querySelector('ul');

liste.remove();