Cours de Développement web 2

Bootstrap & JavaScript

Prise en main de bootstrap

- Vous créez des pages web et vous passez beaucoup de temps avec le CSS?

Un framework CSS, qu'est-ce que c'est ?

- Un framework est avant tout un cadre de travail permettant de faire profiter aux développeurs d'une arborescence et d'une architecture communes.
- Le plus souvent cela se présente sous forme de dossier pour faciliter l'organisation du code et simplifier la maintenance dans le temps.
- Un framework peut aussi apporter un socle de fonctionnalités par défaut.
- Qu'est-ce qu'on obtient concrétement lorsqu'on télécharge bootstrap ? juste un dossier contenant quelques fichiers css et js.

Bootstrap, c'est quoi ?

- Bootstrap est un framework CSS permettant de faciliter l'écriture de la mise en forme CSS d'un site web.
- Avec Bootstrap, vous gagnerez du temps et n'aurez plus besoin de vous prendre la tête sur un décalage de quelques pixels sur votre page web.
- Tout le CSS a déjà été pensé pour vous. Plus généralement, nous pouvons dire que c'est un rassemblement de différents morceaux de code utiles à la création d'une page web, aussi bien dans la mise en forme CSS que dans l'animation et l'interactivité du contenu JavaScript.

Naissance de Bootstrap

- Bootstrap a été écrit par deux développeurs Twitter pour répondre à des problématiques communes et gagner du temps dans la mise en forme des pages web.
- Mark Otto et Jacob Thornton ont écrit bootstrap aux alentours de l'année 2010 puisque le projet a été rendu OpenSource en 2011.
- Cela permet notamment de ne pas devoir recommencer toute l'écriture du code de zéro à chaque nouvelle page web.

Avantages de Bootstrap

- « Bootstrap, j'en entends parler partout, okay mais quel est son intérêt ? je suis très bien sans lui, pourquoi je devrai l'utiliser ? »
- Selon moi, les principaux avantages de bootstrap sont :

1. Responsive : Vous obtiendrez un site web responsive de base, ce qui n'est pas négligeable, cela est même indispensable de nos jours avec les smartphones et tablette !

2. Un site web jolie : Il existe plusieurs thèmes préfaits de haute qualité et une multitude de classes css que l'on peut utiliser !

3. Un gain de temps : Vous ne vous occupez quasiment plus que d'écrire le code Html et d'appeler des classes CSS pré-existantes. Il devient très rapide de monter la mise en forme d'une page web.

4. Compatibilité : A priori, la présentation visuelle sera la même sur tous les navigateurs.

5. Graphisme : Lorsqu'on développe en programmation, cela nous permet d'avoir une apparence de site sans écrire une seule ligne de CSS (à l'origine, bootstrap a justement été créé pour simplifier le travail de présentation visuelle aux développeurs côté back). Cela est une aubaine pour les développeurs qui préfèrent la partie fonctionnelle à la partie graphique.

Naturellement, cette liste n'est pas exhaustive, il y a d'autres avantages, comme notamment l'intégration de plugin jQuery, ce qui permet de gagner également du temps sur la partie JavaScript.

Toutes les versions de bootstrap

20 VERSIONS DEPUIS SEPTEMBER 20, 2011:

4.3.1 - February 13, 2019 (133 ko)
4.3.0 - February 11, 2019 (129 ko)
4.2.1 - December 25, 2018 (126 ko)
4.1.3 - July 25, 2018 (119 ko)
4.1.2 - July 23, 2018 (118 ko)
4.1.1 - April 30, 2018 (117 ko)
4.1.0 - April 16, 2018 (117 ko)
4.0.0 - January 18, 2018 (114 ko)
4.0.0.beta3 - December 29, 2017 (114 ko)
4.0.0.beta2.1 - October 23, 2017 (112 ko)
4.0.0.beta2 - October 19, 2017 (112 ko)
4.0.0.beta - August 11, 2017 (117 ko)
4.0.0.alpha6 - January 07, 2017 (116 ko)
4.0.0.alpha5 - October 22, 2016 (114 ko)
4.0.0.alpha4 - September 10, 2016 (113 ko)
4.0.0.alpha3.1 - July 31, 2016 (113 ko)
4.0.0.alpha3 - December 15, 2015 (111 ko)
4.0.0.alpha2 - November 23, 2015 (110 ko)
4.0.0.alpha1 - November 02, 2015 (111 ko)
0.0.1 - September 20, 2011 (3,5 ko) retiré

Origine de Bootstrap

- Vous connaissez forcément Twitter, un des principaux réseaux sociaux qui inondent la planète de liens virtuels entre les humains devenus des noyaux cybernétiques. Le projet Bootstrap a été créé au départ par Mark Otto et Jacob Thornton pour répondre à des besoins internes de développement de cette entreprise au niveau de l'uniformisation de l'aspect des pages web. Il s'agissait juste de stylisation CSS, mais le framework s'est ensuite enrichi de composants Javascript.
- Il a ensuite été publié en 2011 en devenant rapidement populaire parce qu'il est venu se positionner dans un espace vacant du développement. Son système de grille de 12 colonnes est devenu une référence. D'autre part sa mise en œuvre est aisée et se limite à référencer quelques librairies, comme nous allons bientôt le voir.
- Il a été mis à disposition du public sous licence Apache. Le framework en est actuellement à la version 3. Celle-ci a pris un virage particulier en intégrant l'aspect « responsive » par défaut, alors qu'auparavant cette fonctionnalité faisait l'objet d'un fichier séparé. Cette version est même déclarée comme « mobile-first ». Avec l'utilisation croissante d'appareils mobiles, le framework s'est adapté pour offrir une solution censée couvrir tous les besoins.

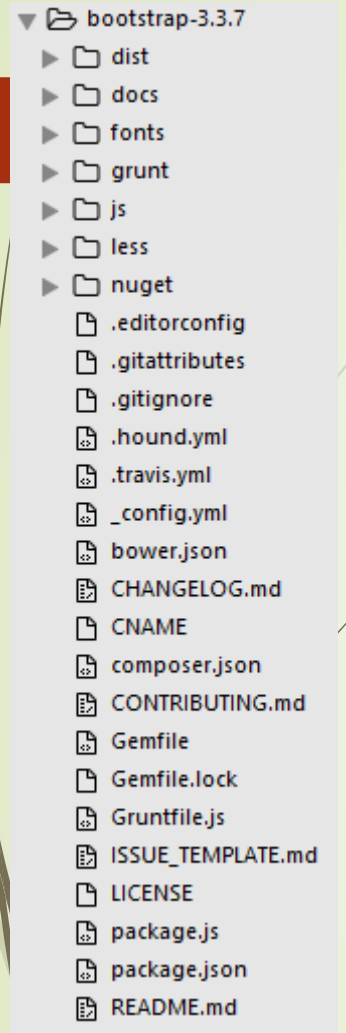
Installation de Bootstrap

- L'installation de Bootstrap est simple : cliquez sur le bouton de téléchargement sur le site du framework. Vous avez à disposition trois boutons :
- "Download Bootstrap" : permet de récupérer juste les fichiers nécessaires au fonctionnement de Bootstrap.
- "Download source" : permet de récupérer en plus tous les fichiers sources.
- "Download Sass" : c'est un portage de Bootstrap en Sass pour les utilisateurs de projets qui utilisent Sass (Rails, Compass...).
- Vous pouvez aussi aller consulter directement le code source sur github, ou même l'installer avec Bower si vous utiliseznode.
- Quand vous téléchargez la librairie avec le bouton "Download source", vous obtenez un fichier zippé contenant un répertoire bootstrap-3.3.7 qui contient lui-même un certain nombre de fichiers et de dossiers, comme le montre la figure suivante.

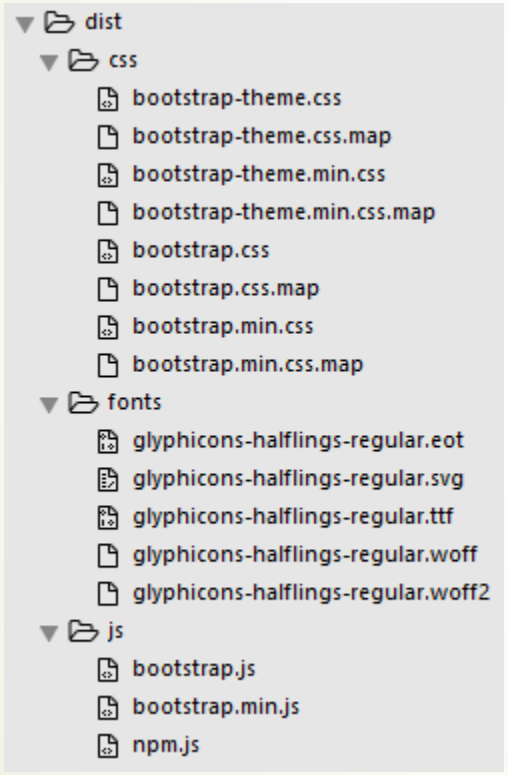
Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.

Download Bootstrap

Currently v3.3.7



Les fichiers utiles pour simplement utiliser Bootstrap se situent dans le dossier`dist`(« distribution »), ce sont les seuls fichiers que vous obtenez si vous utilisez le bouton "Download Bootstrap".



Le dossier « dist » contient les fichiers utiles à l'utilisation de Bootstrap
Voyons un peu les principaux fichiers :

- bootstrap.csscomporte les classes de base de Bootstrap ;
- bootstrap.min.csscomporte les mêmes classes de base quebootstrap.cssmais est minifié ;
- bootstrap-theme.csscontient des règles de styles particulières pour créer un thème spécifique pour Bootstrap ;
- bootstrap-theme.min.cssest identique àbootstrap-theme.cssmais est minifié ;
- glyphicons-halflings-regular.svgcomporte la collection d'icônes au format svg ;
- glyphicons-halflings-regular.ttfcomporte la collection d'icônes au format ttf ;
- glyphicons-halflings-regular.woffcomporte la collection d'icônes au format woff ;
- glyphicons-halflings-regular.eotcomporte la collection d'icônes au format eot ;
- bootstrap.jscontient le code JavaScript des composants de Bootstrap ;
- bootstrap.min.jscontient le même code JavaScript mais est minifié ;
- Les fichiersmin(bootstrap.min.css,bootstrap.min.jsetbootstrap-theme.min.css) contiennent le même code que leurs équivalents (bootstrap.css,bootstrap.jsetbootstrap-theme.css) mais ont été épurés des commentaires et compressés pour les alléger et accélérer ainsi leur chargement.

Définir la taille d'un élément relativement à celle de son parent

- Bootstrap va déjà nous permettre de définir la taille de nos éléments en fonction de celle de leur parent. Pour modifier la largeur d'un élément, on va pouvoir utiliser les classes suivantes :
- **.w-25** : l'élément a une largeur égale à 25% de celle de son parent ;
- **.w-50** : l'élément a une largeur égale à 50% de celle de son parent ;
- **.w-75** : l'élément a une largeur égale à 75% de celle de son parent ;
- **.w-100** : l'élément a une largeur égale à celle de son parent ;
- **.w-auto** : la largeur de l'élément est définie automatiquement.

Pour modifier la hauteur d'un élément, on utilisera les classes suivantes :

- **.h-25** : l'élément a une hauteur égale à 25% de celle de son parent ;
- **.h-50** : l'élément a une hauteur égale à 50% de celle de son parent ;
- **.h-75** : l'élément a une hauteur égale à 75% de celle de son parent ;
- **.h-100** : l'élément a une hauteur égale à celle de son parent ;
- **.h-auto** : la hauteur de l'élément est définie automatiquement.

Ajouter des marges aux éléments avec Bootstrap

- Bootstrap met à notre disposition une série de classes qui vont nous permettre de définir des marges intérieures (le padding) ou extérieures (margin) tout en conservant l'aspect responsive de notre page

Utiliser les notations de padding ou de margin de Bootstrap

- Les classes Bootstrap nous permettant d'appliquer des marges intérieures ou extérieures à nos éléments vont toutes être construites sur le même modèle qui est le suivant : {type de marge}{côté}-{taille}.
- Dans le cas où l'on souhaite appliquer des marges différentes à nos éléments selon la taille de l'écran de nos visiteurs (marges responsive), nous devrons également ajouter un breakpoint entre les notations relatives au côté d'application de la marge et sa taille. Le schéma de la classe sera alors le suivant : {type de marge}{côté}-{breakpoint}-{taille}.
- Pour le premier élément de notre classe, c'est-à-dire le type de marge, nous allons pouvoir choisir entre deux valeurs :

- m va nous servir à définir une marge extérieure (margin) ;
- p va nous servir à définir une marge intérieure (padding).

- Au niveau du côté de l'élément auquel doit s'appliquer la marge, nous avons le choix entre différentes valeurs :

- t va nous permettre d'appliquer une marge (margin ou padding) au niveau du côté supérieur à un élément. Cela va donc servir à définir une margin-top ou un padding-top ;
- b va nous permettre d'ajouter une marge (margin ou padding) basse à un élément ;
- r va nous permettre d'ajouter une marge (margin ou padding) droite à un élément ;
- l va nous permettre d'ajouter une marge (margin ou padding) gauche à un élément ;
- x va nous permettre de définir des marges (margin ou padding) gauche et droite à notre élément ;
- y va nous permettre de définir des marges (margin ou padding) haute et basse à notre élément.

Notez que si on omet le paramètre « côté » dans notre classe, alors la marge s'appliquera à tous les côtés de l'élément à la fois.

- En termes de breakpoint, nous allons pouvoir utiliser les notations habituelles à savoir sm, md, lg et xm. Notre ici que si on ne précise pas de breakpoint alors cela revient à définir une marge pour un breakpoint xs c'est-à-dire pour toutes les tailles d'écran.

Finalement, nous allons pouvoir choisir parmi 7 valeurs différentes pour définir la taille de nos marges :

- 0 va supprimer toutes les marges (margin ou padding) appliquées à un élément ;
- 1 va définir des marges (margin ou padding) de 0.25rem par défaut (0,25 * la valeur de la variable SASS $spacer qui est définie par défaut à 1rem) ;
- 2 va définir des marges (margin ou padding) de 0.5rem par défaut (0.5 * $spacer);
- 3 va définir des marges (margin ou padding) de 1rem par défaut (1 * $spacer);
- 4 va définir des marges (margin ou padding) de 1.5rem par défaut (1.5 * $spacer);
- 5 va définir des marges (margin ou padding) de 3rem par défaut (3 * $spacer);
- La valeur auto va être exclusive aux marges extérieures et définir une margin : auto.

Exemples

Pas de marge

padding-left: 1rem

margin-left: 1rem

padding-left: 1rem et margin-left: 1rem

padding: 0.5rem

Ajouter une bordure classique à un élément en utilisant Bootstrap

Nous allons déjà pouvoir ajouter une bordure carrée autour d'un élément ou sur un côté spécifique d'un élément. Pour cela, nous allons utiliser les classes suivantes :

- **.border** : va ajouter une bordure de couleur grise par défaut tout autour de l'élément ;
- **.border-top** : va ajouter une bordure de couleur grise par défaut uniquement sur le côté haut de l'élément ;
- **.border-right** : va ajouter une bordure de couleur grise par défaut à droite uniquement de l'élément ;
- **.border-bottom** : va ajouter une bordure de couleur grise par défaut uniquement sur le côté bas de l'élément ;
- **.border-left** : va ajouter une bordure de couleur grise par défaut à gauche uniquement de l'élément.

Enlever certaines bordures d'un élément

Bootstrap nous propose une fonctionnalité relativement intéressante qui va être de pouvoir supprimer toutes les bordures d'un élément ou seulement une bordure d'un côté en particulier.

Nous allons pouvoir faire cela en ajoutant « 0 » en fin de classe border-*. Cela va donc nous permettre de gérer très simplement quelles bordures doivent s'afficher autour d'un élément.

Pour cela, nous allons utiliser les classes suivantes :

- .border-0 : supprime toutes les bordures de l'élément ;
- .border-top-0 : supprime la bordure supérieure de l'élément ;
- .border-right-0 : supprime la bordure droite de l'élément ;
- .border-bottom-0 : supprime la bordure inférieure de l'élément ;
- .border-left-0 : supprime la bordure gauche de l'élément.

Créer des bordure de couleur avec Bootstrap

Nous allons pouvoir réutiliser le système de couleurs contextuelles de Bootstrap afin de personnaliser la couleur de nos bordures. Les classes disponibles dans la thème de base sont les suivantes :

- .border-primary : la bordure créée sera colorée en bleu ;
- .border-secondary : la bordure créée sera colorée en gris foncé ;
- .border-success : la bordure créée sera colorée en vert ;
- .border-danger : la bordure créée sera colorée en rouge ;
- .border-warning : la bordure créée sera colorée en jaune ;
- .border-info : la bordure créée sera colorée en une autre teinte de bleu ;
- .border-light : la bordure créée sera colorée en gris clair ;
- .border-dark : la bordure créée sera colorée en noir ;
- .border-white : la bordure créée sera colorée en blanc.

Notez bien que ces classes ne vont pas créer de bordures mais simplement ajouter une couleur à une bordure déjà existante. Il va donc également falloir utiliser une classe border-* pour avant tout créer des bordures pour nos éléments.

Créer des bordures arrondies avec Bootstrap

Nous allons finalement pouvoir changer le comportement et notamment arrondir des bordures avec Bootstrap en utilisant les classes suivantes. Notez bien encore une fois que ces classes ne vont pas créer de bordures mais simplement modifier l'apparence d'une bordure déjà existante. Pour créer notre bordure, nous devrons également utiliser une classe border-*.

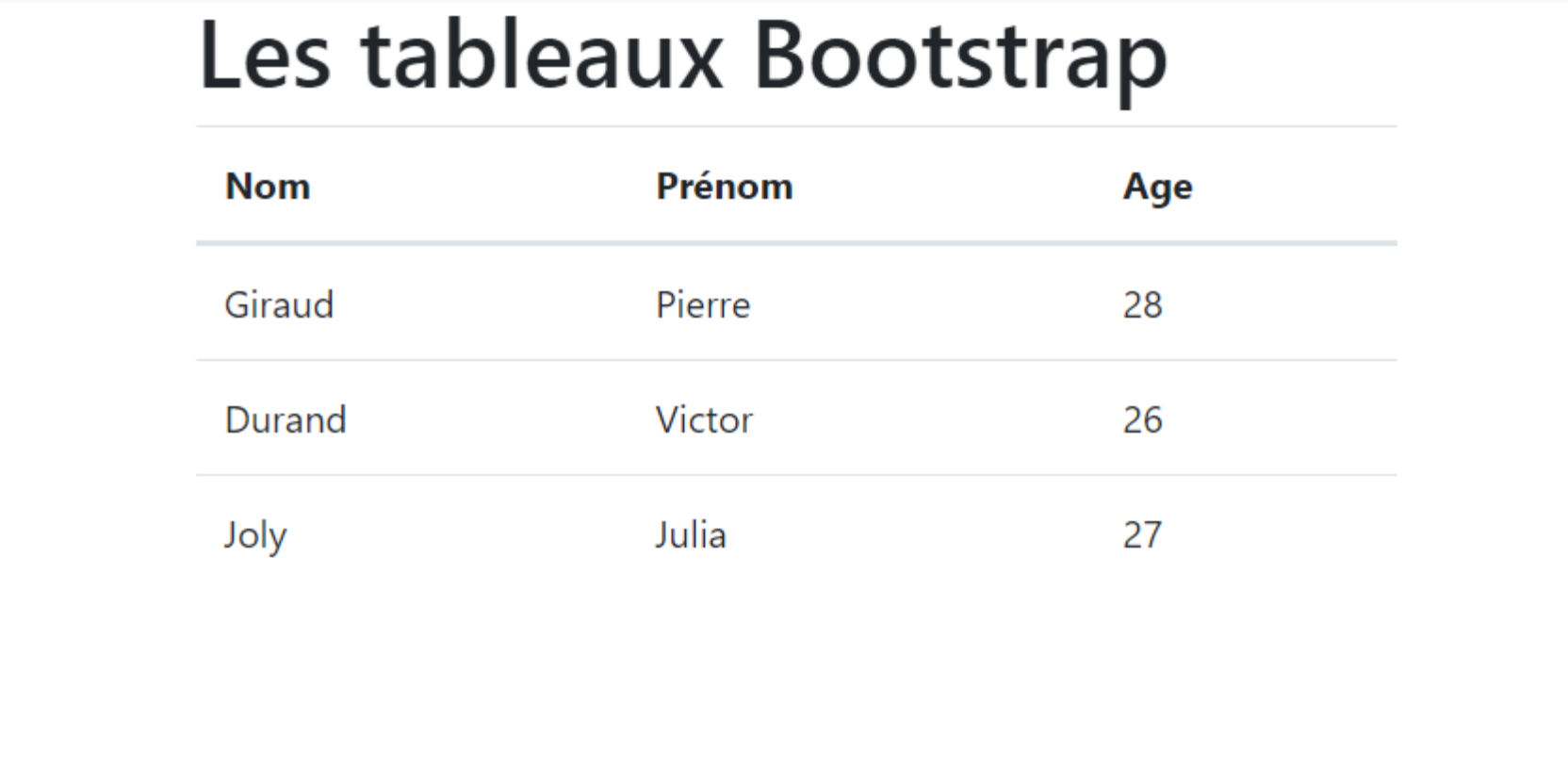
- rounded : arrondit légèrement une bordure ;
- rounded-top : arrondit légèrement les coins supérieurs d'une bordure ;
- rounded-right : arrondit légèrement les coins droits d'une bordure ;
- rounded-bottom : arrondit légèrement les coins inférieurs d'une bordure ;
- rounded-left : arrondit légèrement les coins gauche d'une bordure ;
- rounded-circle arrondit complètement la bordure ;
- rounded-pill : arrondit la bordure jusqu'à obtenir un demi-cercle ;
- rounded-0 : annule l'arrondi d'une bordure ;

Notez qu'on va également pouvoir exercer un certain contrôle sur l'importance de l'arrondi des bordures avec les classes .rounded-sm et .rounded-lg qui vont nous permettre de définir des arrondis moins prononcés ou au contraire plus prononcés.

Créer des tableaux stylisés avec Bootstrap

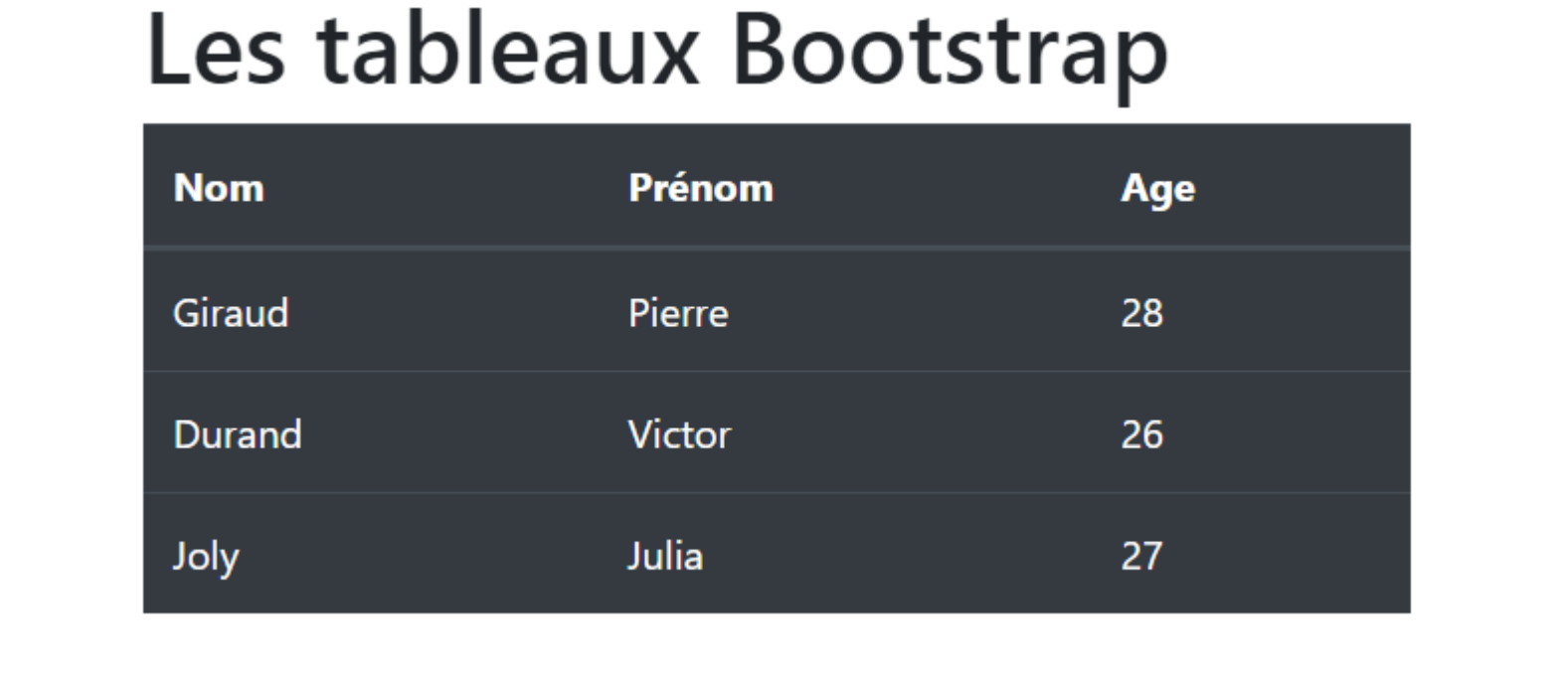
- La classe .table

La classe .table est la classe Bootstrap de base pour styliser des tableaux. Nous allons passer cette classe à un élément table. Celle-ci va appliquer une première mise en forme relativement basique à notre tableau.



Inverser les couleurs d'un tableau

- On va pouvoir inverser les couleurs par défaut d'un tableau Bootstrap, c'est-à-dire avoir une couleur de fond noire et une couleur de texte blanche en ajoutant la classe .table-dark en plus de la classe .table à notre élément HTML table.



Personnaliser l'en-tête d'un tableau

- Bootstrap met à notre disposition deux classes qu'on va pouvoir appliquer à un élément thead pour personnaliser l'en-tête de nos tableaux : les classes .thead-light (ligne d'en-tête avec un fond gris clair) et .thead-dark (ligne d'en-tête avec un fond noir).

Les tableaux Bootstrap

Nom	Prénom	Age
Giraud	Pierre	28
Durand	Victor	26
Joly	Julia	27

Nom	Prénom	Age
Giraud	Pierre	28

Créer un tableau avec alternance de couleurs entre les lignes

- La classe Bootstrap .table-striped qu'on va pouvoir appliquer à notre élément table va nous permettre de créer des tableaux zébrés avec une ligne au fond blanc et une ligne au fond gris clair en alternance.

- Nous pouvons utiliser cette classe avec la classe .table-dark afin de créer un tableau zébré foncé avec alternance de lignes au fond noir et gris foncé.

Ajouter ou supprimer des bordures d'un tableau

- Nous allons pouvoir ajouter des bordures autour de chaque cellule et autour de notre tableau grâce à la classe .table-bordered. Nous allons une nouvelle fois appliquer cette classe à notre élément table.
- Nous allons bien évidemment pouvoir combiner cette classe aux classes .table-dark et .table-striped.
- Pour au contraire supprimer toutes les bordures d'un tableau, on va pouvoir utiliser la classe .table-borderless.

Les tableaux Bootstrap

Nom	Prénom	Age
Giraud	Pierre	28
Durand	Victor	26
Joly	Julia	27

Nom	Prénom	Age
Giraud	Pierre	28
Durand	Victor	26
Joly	Julia	27

Ajouter des effets lors du passage de la souris

- Nous allons encore pouvoir ajouter un peu d'interactivité à notre tableau en changeant la couleur de fond d'une ligne lors du survol de la souris de celle-ci en appliquant la classe .table-hover à notre élément table.
- Cette classe va à nouveau pouvoir être combinée aux précédentes pour créer des effets intéressants.
- Notez que seules les lignes à l'intérieur de l'élément tbody vont changer de couleur de fond au survol de la souris.

Utiliser les couleurs contextuelles avec les tableaux

- Nous allons pouvoir utiliser les couleurs contextuelles de Bootstrap pour changer la couleur de chaque ligne ou de chaque cellule d'un tableau. On va pouvoir utiliser les classes suivantes :

- **.table-active** ;
- **.table-primary** ;
- **.table-secondary** ;
- **.table-success** ;
- **.table-danger** ;
- **.table-warning** ;
- **.table-info** ;
- **.table-light** ;
- **.table-dark**.
- Notez que ces couleurs ne vont pas fonctionner avec un tableau possédant une classe .table-dark. Un hack connu va alors consister à utiliser plutôt les classes de couleurs contextuelles .bg-* pour arriver à des résultats similaires.

Créer une liste de liens ou de boutons avec Bootstrap

- Bootstrap va nous permettre de créer des listes de boutons ou de liens ou plus exactement de styliser des groupes de boutons ou de liens afin de simuler l'apparence d'une liste.

- Pour cela, nous allons simplement devoir utiliser un élément conteneur pour notre groupe de boutons ou de liens et lui attribuer une classe .list-group.

- Ensuite, nous allons devoir réutiliser la classe .list-group-item pour chaque élément de notre « liste ».

- Notez qu'on va également pouvoir ajouter une classe .list-group-item-action à nos différents liens ou boutons afin que ceux-ci changent d'apparence lors d'un changement d'état de ces éléments (passage de souris, lien ou bouton cliqué, etc.).

- Note : dans le cas d'une liste de boutons, n'utilisez pas de classe .btn.

Créer un menu de navigation avec Bootstrap

- La classe .nav est la classe Bootstrap de base pour la navigation. Cette dernière est construite avec le flexbox et va par défaut appliquer les styles suivants :

```
.nav {
  display: flex;
  flex-wrap: wrap;
  padding-left: 0;
  margin-bottom: 0;
  list-style: none;
}
```

- Nous allons appliquer la classe .nav à l'élément représentant notre barre de navigation. Ce sera généralement un élément de liste ul. On pourra entourer cet élément ul d'un élément nav pour indiquer que le composant créé est bien un composant de navigation.

- On va ensuite pouvoir ajouter les classes .nav-item et .nav-link à nos éléments li (éléments ou onglets de navigation) et a (lien de navigation) pour styliser ces éléments.

Aligner notre menu de navigation

- La classe .nav s'appuie sur le modèle des boites flexibles. On va donc pouvoir aligner nos éléments de navigation horizontalement en utilisant les classes .justify-content-center et .justify-content-end.

Gérer les images et les figures avec Bootstrap

- Bootstrap va nous permettre de centrer des images, d'ajouter des cadres de différentes formes ou encore de de faire en sorte que la taille de nos images s'adapte en fonction de celle de leur élément parent.

Créer des cadres de formes différentes autour des images

- Les classes **.rounded** et **.rounded-circle** vont nous permettre respectivement de rogner notre image de façon à ce que sa forme soit légèrement arrondie ou de la transformer en une ellipse tandis que la classe **.img-thumbnail** va ajouter une bordure visible à une image.

Aligner ou centrer une image

- Nous allons pouvoir faire flotter une image à gauche ou à droite avec les classes .**float-left** et **.float-right**.

Exemple

- Faites bien attention cependant à l'impact de la propriété float sur le design général de vos pages.

Pour centrer une image, on va avoir deux solutions possibles. On va déjà pouvoir créer un élément conteneur et utiliser la classe .text-center. On va également pouvoir changer le type d'affichage de notre image avec la classe .d-block (pour « display : block ») et lui appliquer une marge auto avec la classe .mx-auto.

Adapter la taille des images à la taille de l'écran de vos visiteurs

- Nous allons pouvoir faire en sorte que nos images soient responsives en leur appliquant la classe .img-fluid.

- Cette classe va appliquer les styles CSS max-width : 100% et height : auto à nos images.

Figure et figcaption

- Bootstrap nous fournit également trois classes pour gérer l'apparence des éléments à l'intérieur d'éléments HTML figure et figcaption.

Ces classes sont les suivantes :

- **.figure ;**
- **.figure-img ;**
- **.figure-caption.**

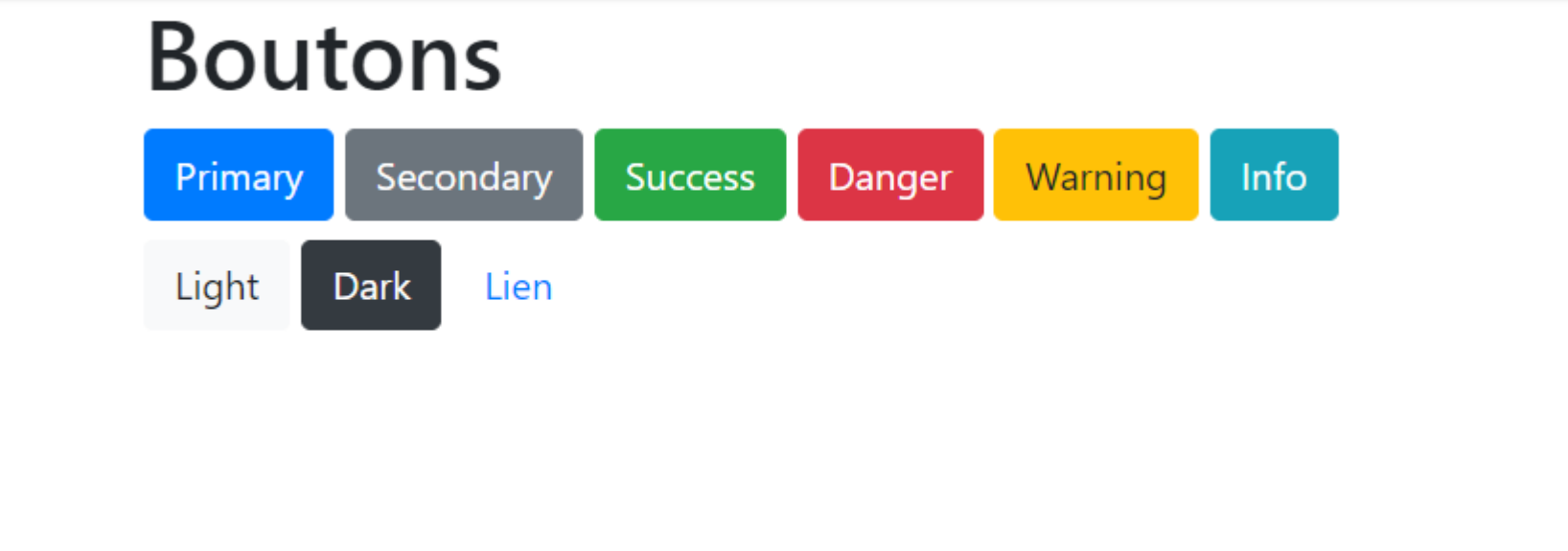
Notez que les images incluses dans les éléments figure n'ont pas de taille prédéfinie. Il faudra donc ajouter ne classe .img-fluid à une image dans cet élément si vous voulez que celle-ci soit responsive.

Créer des boutons stylisés avec Bootstrap

- Bootstrap va nous permettre de styliser des boutons (élément HTMLbutton). Nous allons par exemple pouvoir modifier la couleur de fond ou la forme d'un bouton, le type d'affichage, la taille et l'état d'un bouton.
- Pour personnaliser l'aspect de nos boutons avec Bootstrap, nous allons utiliser la classe de base .btn et des classes de type .btn-*.
- Vous pouvez déjà noter que les classes .btn vont également fonctionner avec des éléments a (liens) et input (champ de formulaire).

Appliquer une couleur de fond à un bouton

- Nous allons pouvoir utiliser les couleurs contextuelles Bootstrap avec nos boutons pour leur appliquer un fond coloré. Lors du survol de la souris sur un bouton, la couleur de fond va légèrement changer.
- Voici les classes que l'on va pouvoir utiliser et leur effet :



- Notez que lorsqu'on travaille avec des boutons en HTML il est considéré comme une bonne pratique de préciser l'intention derrière l'utilisation du bouton avec un attribut type.

- Un bouton défini sans attribut type agira comme si on lui avait passé un type="submit", ce qui signifie qu'il essaiera d'envoyer les données d'un formulaire. Dans notre cas, on précisera type="button" pour expliciter le fait que le bouton n'a pas de comportement spécial par défaut.

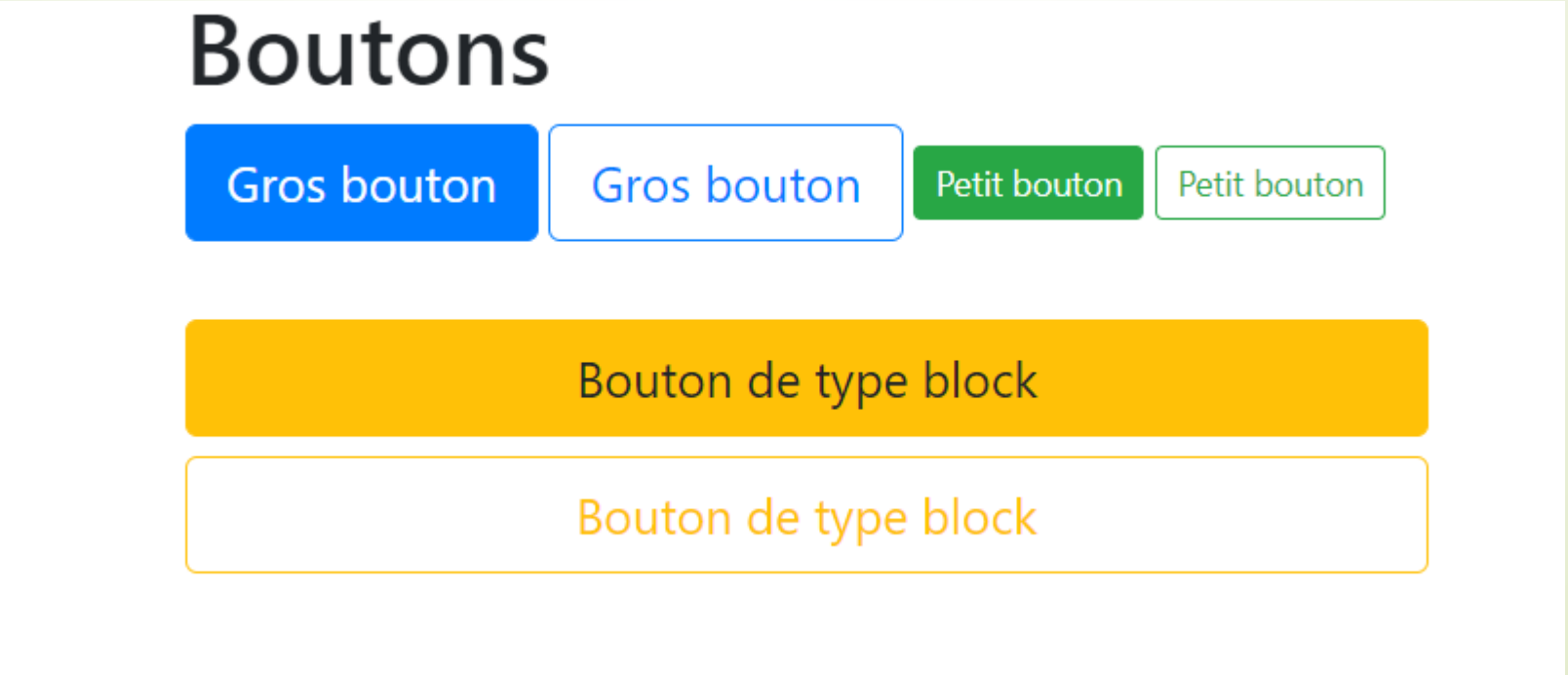
Gérer les bordures des boutons

- Bootstrap nous donne également la possibilité de créer des boutons avec des bordures colorées et un fond blanc.
- Pour cela, nous allons utiliser les classes .btn-outline-* avec les couleurs contextuelles.
- Lors d'un hover, c'est-à-dire lorsque la souris va passer sur un bouton, on va avoir un effet de toggle, c'est-à-dire une inversion des couleurs des différents éléments du bouton : le fond va se colorer et le texte redevenir blanc ou noir selon la couleur de fond du bouton.

Modifier la taille et le type d'affichage d'un bouton

- Bootstrap va également nous permettre de créer des boutons plus petits ou plus grands que la taille par défaut. Pour cela, on va utiliser les classes .btn-sm (bouton de petite taille) et .btn-lg (bouton de grande taille).

- Nous allons également pouvoir changer le type de display d'un bouton et le transformer en élément de type block grâce à la classe .btn-block.

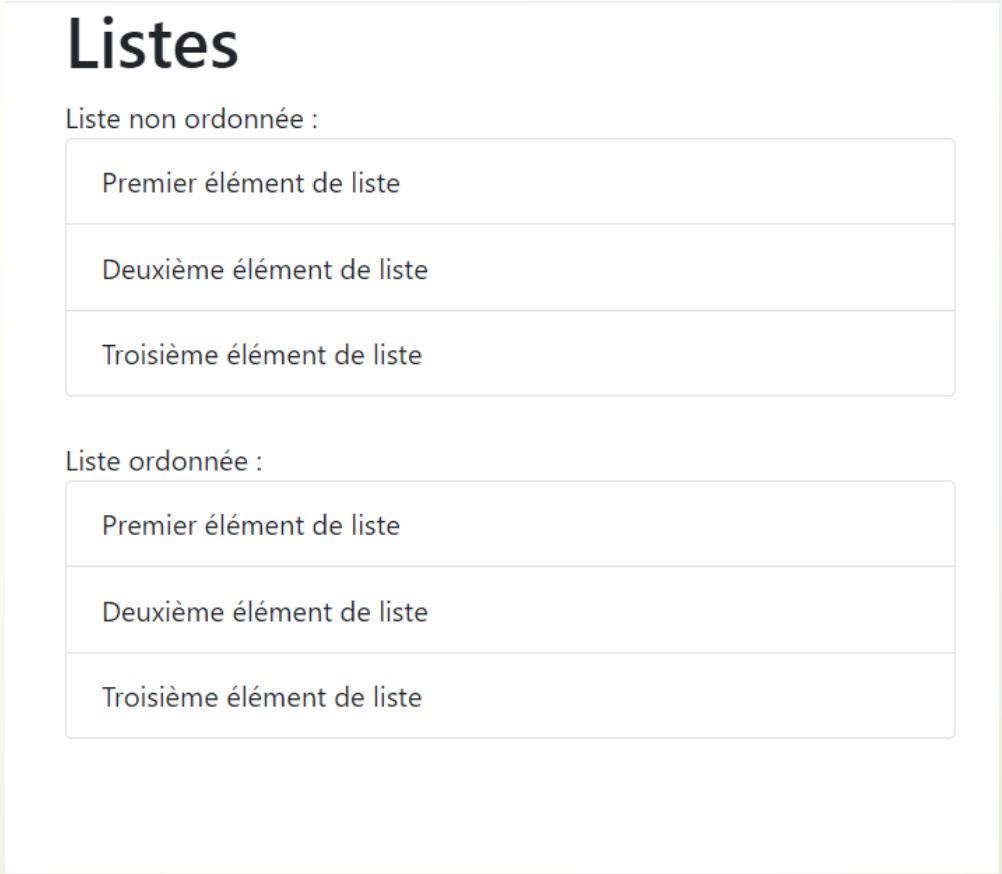


Créer des groupes de listes et appliquer des styles à des listes avec Bootstrap

- Bootstrap nous permet d'ajouter des styles à nos listes et de créer des groupes de listes ou « list groups » qui correspondent à des ensembles d'éléments affichés sous forme de liste.

Les classes .list-group et .list-group-item et la mise en forme de listes HTML

- Pour appliquer une première mise en forme relativement basique à nos listes HTML, on va pouvoir utiliser les classes .list-group sur l'élément représentant la liste et .list-group-item sur les éléments de la liste.



Supprimer les bordures des listes

- Nous allons pouvoir supprimer les bordures autour d'une liste Bootstrap en ajoutant la classe .list-group-flush à notre élément représentant la liste en soi.

Exercice :

- 1- Créer une table de 10 zébré avec une entête (Nom, prénom, adresse, téléphone)
- 2 – Faire un liste non ordonnée sans bordure de 15 lignes
- 3 – faire une liste ordonnée avec bordure et des couleurs contextuelles
- 4 – Utiliser les class btn-sm .btn-lg .btn-block pour créer trois boutons
- 5 - faire flotter une image à gauche et à droite
- 6 -Afficher 4 images en utilisant les class row et col
- 7 – Créer tableau avec un effet au passage de la souris

Créer des listes horizontales

- On va pouvoir changer la présentation de nos listes en faisant en sorte que les éléments de celles-ci s'affichent en ligne plutôt qu'en colonne en utilisant la classe .list-group-horizontal sur l'élément représentant la liste.

- Notez que des variantes responsive de cette classe existent avec .list-group-horizontal-{sm | md | lg | xl} qui permettent de modifier le comportement par défaut d'une liste pour que ses éléments s'affichent en ligne à partir d'une certaine taille de fenêtre. Ces tailles de transition ou « breakpoints » sont les mêmes que d'habitude à savoir 576px pour sm, 768px pour md, 992px pour lg et 1200px pour xl.

Structurer et styliser des formulaires avec Bootstrap

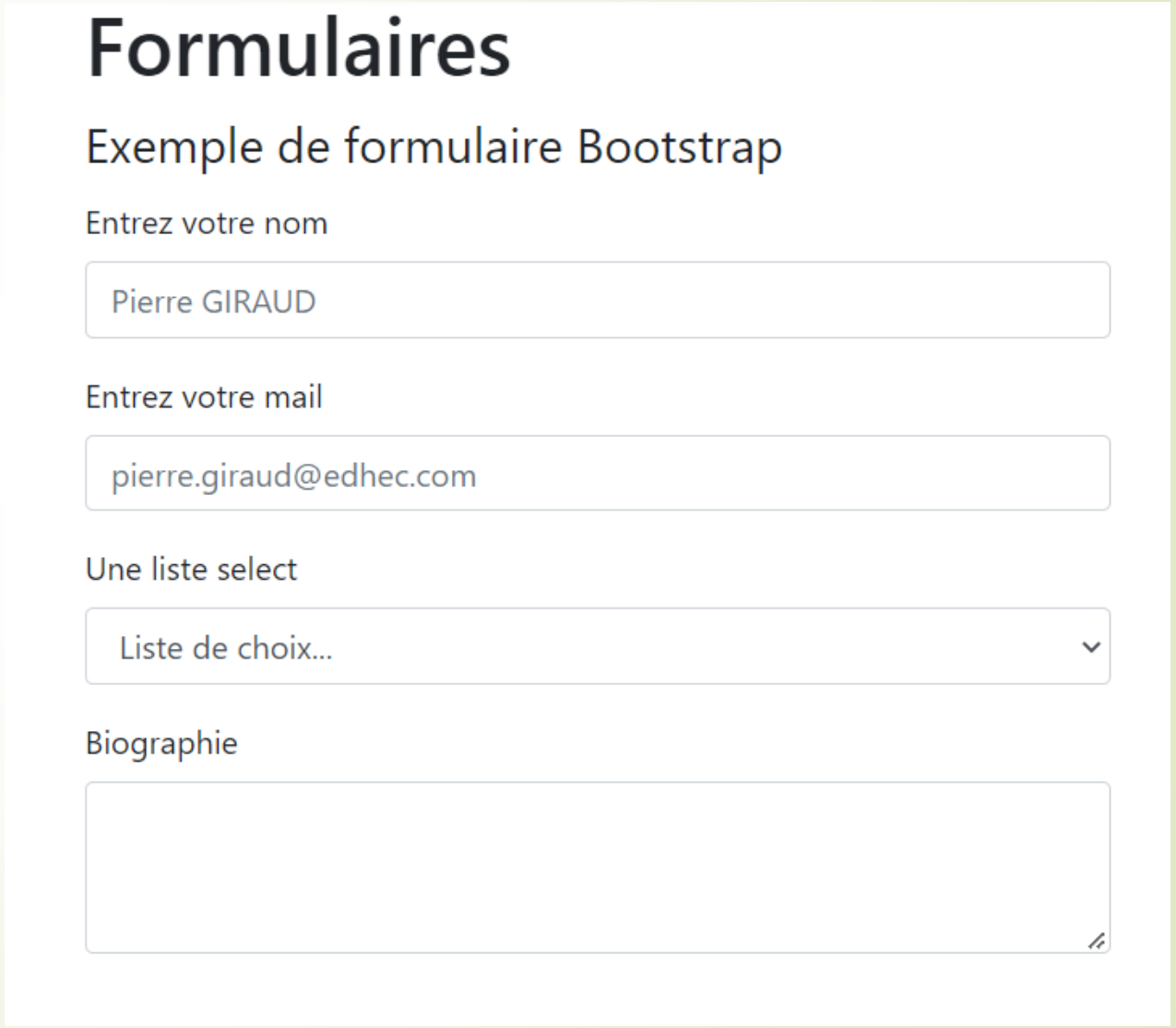
- Bootstrap nous offre de nombreuses classes pour nous permettre de structurer nos formulaires et d'ajouter des styles aux différents éléments des formulaires.

Styliser les champs de formulaire

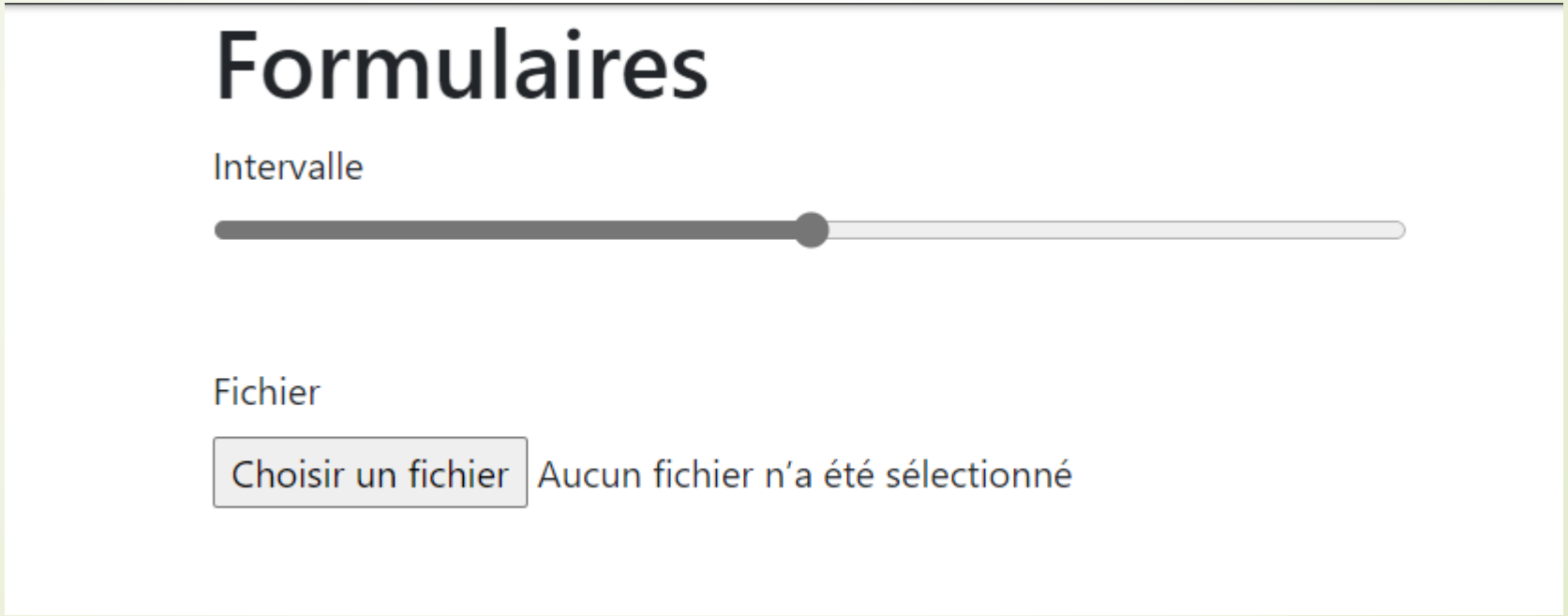
- Bootstrap applique un display : block à la plupart des éléments de formulaire par défaut, ce qui signifie que la plupart des éléments occuperont leur propre ligne.

- On va utiliser la classe **.form-control** pour mettre en forme la majorité des champs de formulaire et notamment les éléments input, select et textarea.

- Nous allons également utiliser une classe **.form-group** pour grouper des label avec le champ de formulaire correspondant. Nous reparlerons de cette classe plus tard.



- Note : on utilisera la classe .form-control-file plutôt que .form-control avec un input type = "file" (champ servant à un envoi de fichier). De même, on utilisera .form-control-range avec un input type = "range" (champ d'intervalle).



Boites d'aide et popovers Bootstrap

- Les boites d'aide ou « tooltips » et les popovers sont des boites qui vont s'afficher soit au survol, soit au clic et fournir des notices d'aide ou de renseignement aux utilisateurs.
- Ces deux composants s'appuient sur le JavaScript et nous allons devoir les initialiser en utilisant du jQuery.

Les boites d'aide ou tooltips

- Les boites d'aide sont des boites de texte qui apparaissent lorsqu'un utilisateur passe sa souris sur un élément ;
- Pour créer une boite d'aide, on ajoutera un attribut data-toggle="tooltip" à un élément. On va utiliser un attribut title pour préciser le texte de notre boite d'aide.
- On va également pouvoir choisir si la boite d'aide doit apparaitre au-dessus, en dessous, à droite ou à gauche de l'élément en précisant un attribut data-placement et en lui passant l'une de valeurs top | right | bottom | left.
- Pour initialiser les boites d'aide, on va devoir sélectionner les éléments portant les boites et appeler la méthode toooltip() avec jQuery.

Apprenez à coder avec JavaScript

- Pour créer des **programmes d'ordinateur** ou bâtir l'application de vos rêves, vous devez savoir comment **coder dans un langage de programmation**. Les applications s'appuient sur des données et une logique pour accomplir leur magie, et cela exige qu'un programmeur – vous – indique à l'ordinateur ce qu'il faut faire et comment. Mais comment écrire des applications, au fait ? Par où commencer ?
- Ce cours est conçu pour vous enseigner les bases du **langage de programmation JavaScript** et pour vous apporter beaucoup de pratique au passage ! Nous nous intéresserons à :

- **Ajoutez des conditions**
- Jusqu'à présent, toutes les instructions de nos programmes (à part les commentaires) étaient systématiquement exécutées. Nous allons voir comment enrichir nos programmes en y ajoutant des possibilités d'exécution conditionnelle.
- **Exprimer une condition**
- Imaginons qu'on souhaite écrire un programme qui fasse saisir un nombre à l'utilisateur, puis qui affiche un message si ce nombre est positif. Voici l'algorithme correspondant.
- ***Saisir un nombre***
- ***Si ce nombre est positif***
- ***Afficher un message***
- L'affichage du message ne doit avoir lieu que si le nombre est positif : on dit qu'il est soumis à une condition.

Déclarez des variables et modifiez leurs valeurs

- L'objectif d'un programme informatique est de faire quelque chose avec des données. Votre programme utilise des **variables** pour enregistrer et manipuler ces données. Plus précisément, une variable est un **contenant** utilisé pour enregistrer une donnée spécifique dont votre programme a besoin pour travailler. Un nom d'utilisateur, le nombre de billets restants pour un vol, la disponibilité ou non d'un certain produit en stock, toutes ces données sont enregistrées dans des variables.

```
let numberOfCats = 2;
let numberOfDogs = 4;
```

Opérateur	Signification
===	Egal à
!==	Différent de
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à

Modifiez la valeur d'une variable

- La façon la plus simple de modifier la valeur d'une variable est simplement de la réaffecter :

let numberOfCats = 3;

numberOfCats = 4;

Opérateurs arithmétiques – travail sur des nombres

- Les opérateurs arithmétiques vous permettent d'effectuer des opérations mathématiques de base telles que l'addition, la soustraction, la multiplication et la division. Toutes ces opérations peuvent être extrêmement utiles, même en dehors des contextes mathématiques.
- **Addition et soustraction**

Pour ajouter deux variables, utilisez le signe **+**

```
let totalCDs = 67;
let totalVinyls = 34;
let totalMusic = totalCDs + totalVinyls;
```

À l'inverse, la soustraction utilise le signe **-**

```
let cookiesInJar = 10;
let cookiesRemoved = 2;
let cookiesLeftInJar = cookiesInJar - cookiesRemoved;
```

- Pour ajouter ou soustraire un nombre d'une variable, vous pouvez utiliser les opérateurs **+=** et **-=**

let cookiesInJar = 10;

/* manger deux cookies */

cookiesInJar -= 2; //il reste 8 cookies

/* cuisson d'un nouveau lot de cookies */

cookiesInJar += 12; // il y a maintenant 20 cookies dans la boîte

Enfin, vous pouvez utiliser **++** ou **--** pour ajouter ou soustraire 1 (incrément ou décrément) :

let numberOfLikes = 10;

numberOfLikes++; // cela fait 11

numberOfLikes--; // et on revient à 10...qui n'a pas aimé mon article ?

- **Multiplication et division**

Les opérations de multiplication et de division utilisent les opérateurs * et / :

```
let costPerProduct = 20;
let numberOfProducts = 5;
let totalCost = costPerProduct * numberOfProducts;
let averageCostPerProduct = totalCost / numberOfProducts;
```

Comme pour l'addition et la soustraction, il existe aussi les opérateurs *= et /= pour multiplier ou diviser un nombre :

```
let numberOfCats = 2;
numberOfCats *= 6;  // numberOfCats vaut maintenant 2*6 = 12;
numberOfCats /= 3;  // numberOfCats vaut maintenant 12/3 = 4;
```

types de données

- **Qu'est-ce qu'un type ?**

Le **type** d'une variable ou d'une constante est tout simplement le genre des données qu'elle enregistre. En JavaScript, il y a trois types primitifs principaux :

- number (nombre) ;
- string (chaîne de caractères) ;
- boolean (valeur logique).

Découvrez le type "number"

- Toutes les variables que vous avez créées jusqu'à maintenant dans ce cours étaient du type number (nombre, en français). Comme vous l'avez vu, elles peuvent être manipulées de nombreuses façons.

- Les variables de type number peuvent être positives ou négatives. Elles peuvent aussi être des nombres entiers (1, 2, 3, etc.) ou décimaux (1,4 ; 67,34 ; etc.).

Appréhendez les valeurs logiques (booleans)

- Les valeurs logiques (booleans) sont le plus simple des types primitifs : elles ne peuvent avoir que deux valeurs, true ou false (vrai ou faux). Elles s'utilisent dans toutes sortes de cas : pour indiquer si un utilisateur est connecté ou non, si une case est cochée ou non, ou si un ensemble de conditions particulières est réuni.

Les variables de type boolean peuvent avoir une des deux valeurs true ou false

```
let userIsSignedIn = true;

let userIsAdmin = false;
```

les chaînes de caractères

- Les chaînes de caractères (chaînes, ou strings, en anglais) sont la façon d'enregistrer du texte dans des variables JavaScript. On peut enregistrer dans une variable de type string n'importe quelle chaîne de caractères, allant d'une seule lettre à un très grand nombre de lettres (plus de 134 millions, même dans des navigateurs anciens).

- Les variables de type string sont encadrées par des guillemets simples ou doubles –' ou " :

let firstName = "Will";

let lastName = 'Alexander';

- **concaténées** : Les chaînes peuvent aussi être concaténées (ajoutées à la fin l'une de l'autre) par l'opérateur +

let wholeName = firstName + " " + lastName; // valeur: "Will Alexander"

Exemple :

const myName = `Alexander`;

const salutation = `Bienvenue sur mon site ${myName}!`;

console.log(salutation); //retournera "Bienvenue sur mon site Alexander!"

Exercice

1. Créez une variable nommée episodeTitle qui contient une chaîne de caractères correspondant au titre de votre épisode (employez votre créativité !).
2. Créez une variable appelée episodeDuration qui contient un nombre correspondant à la durée de votre épisode en minutes.
3. Enfin, créez une variable booléenne nommée hasBeenWatched qui indique si l'utilisateur a déjà regardé l'épisode ou non.

Vérifiez si le composant fonctionne correctement. N'hésitez pas à modifier vos valeurs pour vérifier que le composant réagit comme prévu. N'oubliez pas, votre code ira entre les commentaires "===" sur l'exercice.

Exercice 2

- Créez une variable nommée ecole qui contient une chaîne de caractères correspondant au nom de votre établissement
- Créez une variable appelée nom_dep qui contient une chaîne de caractères correspondant au nom de votre département
- Créez une variable appelée nom_prenom qui contient une chaîne de caractères correspondant a votre nom et prénom
- Créez une variable appelée telephone qui contient votre numéro de téléphone.
- Créez une variable nommée infos qui correspont (nom_prenom+nom_dep+ ecole+telephone

L'instruction if

- Voici comment ce programme se traduit en JavaScript.

```
const nombre = Number(prompt("Entrez un nombre :"));
if (nombre > 0) {
  console.log(nombre + " est positif");
}
```

Vous venez de découvrir comment soumettre l'exécution d'une partie d'un programme à une condition grâce à l'instruction JavaScript If. La syntaxe de cette instruction est la suivante.

```
if (condition) {
  // instructions exécutées quand la condition est vraie
}
```

La paire d'accolades ouvrante et fermante délimite ce que l'on appelle un **bloc de code** associé à l'instruction IF. Cette instruction représente un **test**. On peut la traduire par l'ordre suivant : "Si la condition est vraie, alors exécute les instructions contenues dans le bloc de code".

La condition est toujours placée entre parenthèses après le if.

Les instructions du bloc de code associé sont décalées vers la droite par rapport au if.

Cette pratique est appelée l'indentation et permet de rendre les programmes bien plus lisibles.

A mesure que vos programmes deviendront plus complexes (avec desif et d'autres instructions étudiées plus loin), leur indentation deviendra essentielle pour faciliter leur lisibilité.

Quelle que soit la valeur d'indentation choisie (entre 2 et 4 espaces), il est indispensable de toujours bien indenter son code !

La notion de condition

- Une **condition** est une expression dont l'évaluation produit une valeur soit vraie, soit fausse : on parle de valeur **booléenne**.
- **Quand la valeur d'une condition est vraie, on dit que cette condition est vérifiée**
- Nous avons déjà étudié les types nombre et chaîne : le type booléen fait également partie des types supportés par le langage JavaScript.
- Ce type n'a que deux valeurs possibles :**true (vrai) etfalse (faux).**

```
if (true) {
  // la condition du if est toujours vraie :
  // les instructions de ce bloc seront toujours exécutées
}
if (false) {
  // la condition du if est toujours fausse :
  // les instructions de ce bloc ne seront jamais exécutées
}
```

Toute expression produisant une valeur booléenne (donc soit vraie, soit fausse) peut être utilisée comme condition dans une instructionif.

Si la valeur de cette expression esttrue, le bloc de code associé auif sera exécuté.

On peut créer des expressions booléennes en utilisant les opérateurs de comparaison regroupés dans le tableau suivant.

L'instruction else

- Enrichissons notre programme d'exemple pour qu'il affiche un message adapté au nombre saisi par l'utilisateur.

```
const nombre = Number(prompt("Entrez un nombre :"));
if (nombre > 0) {
  console.log(nombre + " est positif");
}
else {
  console.log(nombre + " est négatif ou nul");
}
```

Une alternative s'exprime en JavaScript grâce à l'instructionelse associée à unif.

Voici sa syntaxe. if (condition) {

// instructions exécutées quand la condition est vraie

}

else {

// instructions exécutées quand la condition est fausse

}

On peut traduire une instructionif/else comme ceci : "Si la condition est vraie, alors exécute les instructions du bloc de code associé auif, sinon exécute celles du bloc de code associé auelse".

L'instructionif/else permet de créer un branchement logique à l'intérieur d'un programme.

Pendant l'exécution, les instructions exécutées seront différentes selon la valeur de la condition. Un seul des deux blocs de code sera pris en compte.

Pour mieux vous familiariser avec le fonctionnement de cette instruction, n'hésitez pas à expérimenter dans l'éditeur ci-dessous.

Imbriquer des conditions

Notre programme d'exemple peut encore être enrichi pour afficher un message spécifique si le nombre saisi est nul.

Pour cela, le code doit être modifié de la manière suivante.

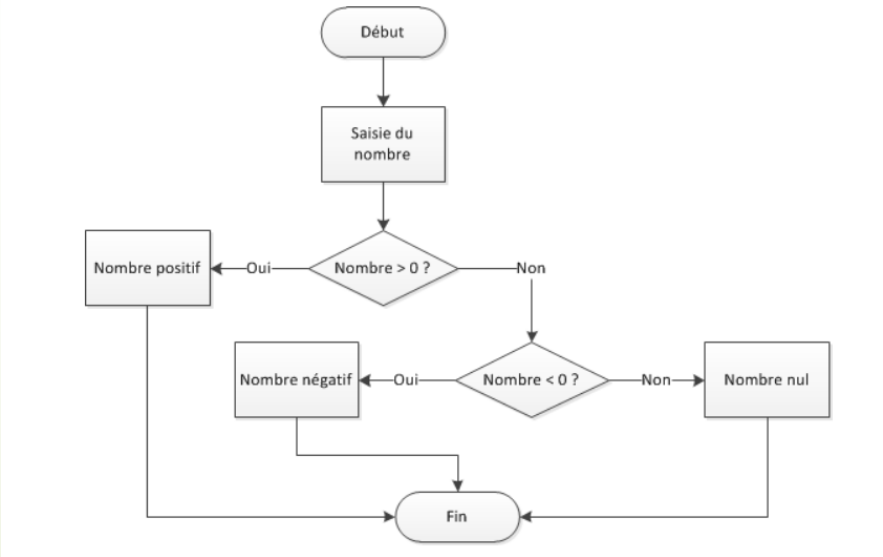
```
const nombre = Number(prompt("Entrez un nombre :"));
if (nombre > 0) {
  console.log(nombre + " est positif");
} else { // nombre <= 0
  if (nombre < 0) {
    console.log(nombre + " est négatif");
  } else { // nombre === 0
    console.log(nombre + " est nul");
  }
}
```

C'est maintenant qu'il faut faire appel à votre sens logique pour le comprendre. Si le premier blocelse est exécuté, c'est que le nombre saisi est soit négatif, soit nul, puisque la condition(nombre > 0) du premierif n'a dans ce cas pas été vérifiée.

A l'intérieur de ce blocelse, on vérifie si le nombre est strictement négatif avec la condition(nombre < 0).

Si cette condition est fausse, alors le nombre est forcément égal à 0.

- C'est maintenant qu'il faut faire appel à votre sens logique pour le comprendre.
- Si le premier blocelse est exécuté, c'est que le nombre saisi est soit négatif, soit nul, puisque la condition(nombre > 0) du premierif n'a dans ce cas pas été vérifiée.
- A l'intérieur de ce blocelse, on vérifie si le nombre est strictement négatif avec la condition(nombre < 0).
- Si cette condition est fausse, alors le nombre est forcément égal à 0.
- Il est possible de représenter graphiquement l'exécution du programme précédent au moyen d'un **diagramme de flux** qui montre les différents cheminements possibles selon la valeur du nombre saisi.



Cet exemple nous montre que l'indentation permet de bien visualiser les différents blocs crées par les instructionsif/else.

Il n'y a pas de limite (si ce n'est la lisibilité du programme) au niveau de profondeur des imbrications.

On rencontre fréquemment le cas particulier où la seule instruction d'un blocelse est unif (leelse éventuellement associé à ceif ne compte pas comme une seconde instruction).

Dans ce cas, il est possible d'écrire ceif sur la même ligne que le premier else, sans accolades ni indentation.

Ainsi, notre programme d'exemple peut être réécrit de la manière suivante.

```
const nombre = Number(prompt("Entrez un nombre :"));
if (nombre > 0) {
  console.log(nombre + " est positif");
} else if (nombre < 0) {
  console.log(nombre + " est négatif");
} else {
  console.log(nombre + " est nul");
}
```

Créer des conditions composées

Exprimer un choix

Essayons d'écrire un programme qui conseille l'utilisateur sur la tenue à porter en fonction de la météo actuelle.

Une première solution consiste à utiliser des instructionsif/else.

```
const meteo = prompt("Quel temps fait-il dehors ?");
if (meteo === "soleil") {
  console.log("Sortez en t-shirt.");
} else if (meteo === "vent") {
  console.log("Sortez en pull.");
} else if (meteo === "pluie") {
  console.log("Sortez en blouson.");
} else if (meteo === "neige") {
  console.log("Restez au chaud à la maison.");
} else {
  console.log("Je n'ai pas compris !");
}
```

- Lorsqu'un programme consiste à déclencher un bloc d'opérations parmi plusieurs selon la valeur d'une expression, on peut l'écrire en utilisant l'instruction JavaScriptswitch.

Voici le programme précédent, réécrit en utilisant un switch.

```
const meteo = prompt("Quel temps fait-il dehors ?");
switch (meteo) {
  case "soleil":
    console.log("Sortez en t-shirt.");
    break;
  case "vent":
    console.log("Sortez en pull.");
    break;
  case "pluie":
    console.log("Sortez en blouson.");
    break;
  case "neige":
    console.log("Restez au chaud à la maison.");
    break;
  default:
    console.log("Je n'ai pas compris !");
}
```

L'instructionswitch déclenche l'exécution d'un bloc d'instructions parmi plusieurs possibles.

Seul le bloc correspondant à la valeur de l'expression testée sera pris en compte.

Sa syntaxe est la suivante.

```
switch (expression) {
  case valeur1:
    // instructions exécutées quand expression vaut valeur1
    break;
  case valeur2:
    // instructions exécutées quand expression vaut valeur2
    break;
  ...
  default:
    // instructions exécutées quand aucune des valeurs ne correspond
}
```

Il n'y a pas de limite au nombre de cas possibles. Le mot-clédefault, à placer en fin deswitch, est optionnel.
Il sert souvent à gérer les cas d'erreurs, comme dans l'exemple ci-dessus.
Les instructionsbreak; dans les blocscase sont indispensables pour sortir duswitch et éviter de passer d'un bloc à un autre.

```
const x = "abc";
switch (x) {
  case "abc":
    console.log("x vaut abc");
    // pas de break : on passe au bloc suivant !
  case "def":
    console.log("x vaut def");
    break;
}
```

L'exécution de cet exemple affiche deux messages :"x vaut abc" (résultat attendu) mais aussi"x vaut def".

La boucle while

La boucle **while** permet de répéter des instructions tant qu'une condition est vérifiée.

Exemple d'utilisation
Voici notre programme d'exemple réécrit avec une bouclewhile.

```
console.log("Début du programme");
let nombre = 1;
while (nombre <= 5) {
  console.log(nombre);
  nombre++;
}
console.log("Fin du programme");
```

Observez le fonctionnement de cet exemple dans l'éditeur ci-dessous.

Fonctionnement
La syntaxe de l'instructionwhile est la suivante.

```
while (condition) {
  // instructions exécutées tant que la condition est vérifiée
}
```

Avant chaque tour de boucle, la condition associée auwhile est évaluée.

Si elle est vraie, les instructions du bloc de code associé auwhile sont exécutées.
Ensuite, l'exécution revient au niveau du while et la condition est à nouveau vérifiée.
Si elle est fausse, les instructions du bloc ne sont pas exécutées et le programme continue juste après le blocwhile.
Le bloc d'instructions associé à une boucle est appelé le corps de la boucle.

Le corps de la boucle doit être placé entre accolades, sauf s'il se réduit à une seule instruction.
Dans un premier temps, je vous conseille d'ajouter systématiquement des accolades à toutes vos boucles.

La boucle for

On a fréquemment besoin d'écrire des boucles dont la condition est basée sur la valeur d'une variable qui est modifiée dans le corps de la boucle, comme dans notre exemple.
Pour répondre à ce besoin, JavaScript et la plupart des autres langages disposent d'un autre type de boucle : le **for**.

Exemple d'utilisation
Voici notre programme d'exemple réécrit avec une bouclefor.

```
let compteur;
for (compteur = 1; compteur <= 5; compteur++) {
  console.log(compteur);
}
```

Testez cet exemple. Vous obtenez exactement le même résultat que précédemment.

Fonctionnement

La syntaxe de l'instruction for est la suivante.

```
for (initialisation; condition; étape) {
    // instructions exécutées tant que la condition est vérifiée
}
```

Son fonctionnement est un peu plus complexe que celui d'unwhile.
Lisez attentivement ce qui suit :
L'initialisation se produit une seule fois, au début de l'exécution.
La condition est évaluée avant chaque tour de boucle. Si elle est vraie, un nouveau tour de boucle est effectué. Sinon, la boucle est terminée.
L'étape est réalisée après chaque tour de boucle.

Le plus souvent, on utilise l'initialisation pour définir la valeur initiale d'une variable qui sera impliquée dans la condition de la boucle.
L'étape sert à modifier la valeur de cette variable.

Compteur de boucle

La variable utilisée dans l'initialisation, la condition et l'étape d'une boucle for est appelée le compteur de la boucle.

Par convention, la variable compteur d'une bouclefor est souvent nomméei.

Très souvent, on n'a pas besoin d'utiliser la variable compteur en dehors du corps de la boucle. Dans ce cas, on peut la déclarer en même temps qu'on l'initialise dans la boucle. Notre programme d'exemple peut être réécrit ainsi :

```
for (let i = 1; i <= 5; i++) {
    console.log(i); // OK
}
console.log(i); // Erreur : la variable i n'est pas visible ici
```

Erreurs fréquentes

Boucle while infinie

Le principal risque lié à la bouclewhile est la "boucle infinie". Il s'agit d'une erreur de programmation très facile à commettre, donc dangereuse.

Modifiez l'exemple de bouclewhile en oubliant volontairement la ligne qui incrémente la variablenombre.

```
let nombre = 1;
while (nombre <= 5) {
    console.log(nombre);
    // La variable n'est plus modifiée : la condition sera toujours vraie
```

Erreurs fréquentes

- Boucle while infinie
- Le principal risque lié à la boucle **while** est la "boucle infinie".
- Il s'agit d'une erreur de programmation très facile à commettre, donc dangereuse.
- Modifiez l'exemple de boucle **while** en oubliant volontairement la ligne qui incrémente la variable **nombre**.

```
let nombre = 1;
while (nombre <= 5) {
    console.log(nombre);
    // La variable n'est plus modifiée : la condition sera toujours vraie
}
```

Lors de l'exécution de ce programme, on effectue un premier tour de boucle puisque la conditionnombre <= 5 est initialement vérifiée.

Mais comme on ne modifie plus la variablenombre dans le corps de la boucle,

la condition est indéfiniment vraie : il s'agit d'une boucle infinie.

- Manipulation du compteur d'une boucle for
- Imaginons qu'un accès de folie temporaire vous conduise à modifier le compteur d'une boucle **for** dans le corps de la boucle, comme dans l'exemple suivant.
- A chaque tour de boucle, la variable **compteur** est incrémentée deux fois :
- dans le corps de la boucle, puis dans l'étape exécutée à la fin de chaque tour.
- ***Quand on emploie une boucle for, la modification du compteur de boucle (le plus souvent une incrémentation) est réalisée à la fin de chaque tour de boucle.***

Sauf exception rarissime, Il ne faut surtout pas modifier le compteur dans le corps de la boucle.

Choix entre un while et un for

Comment choisir le type de boucle à utiliser lorsqu'on doit répéter des instructions dans un programme ?

La bouclefor a l'avantage d'intégrer la modification du compteur dans sa syntaxe, ce qui élimine le problème des boucles infinies.

En revanche, son utilisation implique que le nombre de tours de boucle soit connu à l'avance.

Les scénarios où le nombre de tours ne peut pas être prévu à l'avance seront plus simples à écrire avec unwhile.

C'est notamment le cas lorsque la boucle sert à contrôler une donnée saisie par l'utilisateur, comme dans cet exemple.

```
let lettre = "";
while (lettre !== "X") {
  lettre = prompt("Tapez une lettre ou X pour sortir : ");
}
```

Le nombre de tours de boucle dépend des valeurs saisies par l'utilisateur : il est imprévisible.

En conclusion, le choix entre un **while** et un for dépend du contexte.

Toutes les boucles peuvent s'écrire avec un **while**.

Si on peut prévoir à l'avance le nombre de tours de boucles à effectuer, la boucle for est le meilleur choix.

Sinon, il vaut mieux utiliser le **while**.

Modularisez votre code grâce aux fonctions

- Dans ce chapitre, vous allez découvrir comment décomposer un programme en sous-parties appelées des fonctions.

Introduction : le rôle des fonctions

- Pour comprendre l'intérêt des fonctions, revenons sur notre algorithme de préparation d'un plat de pâtes issu du chapitre d'introduction.

Début
Sortir une casserole
Mettre de l'eau dans la casserole
Mettre la casserole sur le feu
Tant que l'eau ne bout pas
Attendre
Verser les pâtes dans la casserole
Tant que les pâtes ne sont pas cuites
Attendre
Verser les pâtes dans une passoire
Remuer la passoire pour faire couler l'eau
Remettre les pâtes dans la casserole
Goûter
Tant que les pâtes sont trop fades
Ajouter du sel
Goûter
Si on préfère le beurre à l'huile
Ajouter du beurre
Sinon
Ajouter de l'huile
Fin

Introduction : le rôle des fonctions

- Voici le même algorithme. écrit d'une manière différente.

Début

Faire bouillir de l'eau

Cuire les pâtes dans l'eau

Egoutter les pâtes

Assaisonner les pâtes

Fin

- La première version détaille toutes les actions individuelles à réaliser. La seconde décompose la recette en sous-étapes regroupant plusieurs actions individuelles. Cette version est plus concise et plus facile à interpréter, mais elle introduit des concepts relatifs au domaine de la cuisine comme *cuire*, *égoutter*, ou *assaisonner*. On peut envisager de réutiliser ces concepts pour réaliser d'autres recettes, par exemple la préparation d'un plat de riz.
- Jusqu'à présent, vos programmes étaient écrits sur le modèle du premier algorithme : des actions individuelles qui s'enchaînent. Vous allez maintenant apprendre à les concevoir sous la forme d'un ensemble de sous-étapes. En JavaScript, ces sous-étapes sont appelées des fonctions.

Découverte des fonctions

- Une **fonction** est un regroupement d'instructions qui réalise une tâche donnée.
- Voici un exemple basique utilisant une fonction.

```
function direBonjour() {
  console.log("Bonjour !");
}


console.log("Début du programme");
direBonjour();
console.log("Fin du programme");
```

Déclaration d'une fonction

- Observons la première partie du programme d'exemple.

```
function direBonjour() {
  console.log("Bonjour !");
}
```

Cet extrait permet de créer une fonction nommée **direBonjour().** Elle n'est constituée que d'une seule instruction qui affiche sur la console le message "Bonjour !".

L'opération de création d'une fonction s'appelle la **déclaration**. Voici sa syntaxe.

```
// Déclaration d'une fonction nommée maFonction
function maFonction() {
  // Instructions de la fonction
}
```

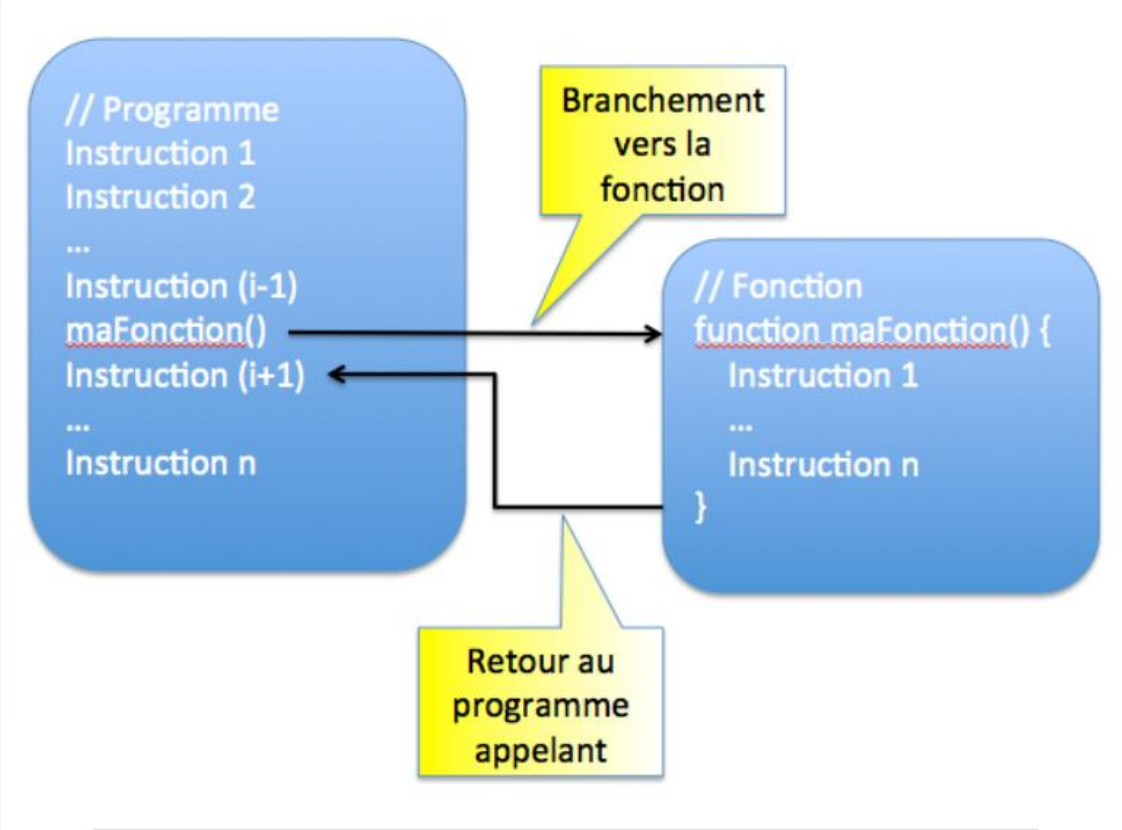
La déclaration d'une fonction s'effectue à l'aide du mot-clé JavaScriptfunction suivi du nom de la fonction et d'une paire de parenthèses. Les instructions qui composent la fonction constituent le corps de la fonction.

Ces instructions sont placées entre accolades et indentées.

L'appel d'une fonction s'effectue en écrivant le nom de la fonction suivi d'une paire de parenthèses.

```
// ...
maFonction(); // Appel de la fonction maFonction
// ...
```

- L'appel d'une fonction déclenche l'exécution des instructions qui la constituent, puis l'exécution reprend à l'endroit où la fonction a été appelée. Ce fonctionnement est illustré par le schéma ci-dessous.



Avantages des fonctions

- Lorsqu'on cherche à résoudre un problème complexe, il est généralement efficace de le décomposer en sous-problèmes plus simples.
- Les fonctions permettent d'appliquer ce principe à la création de logiciels : on va décomposer le programme en écrivant plusieurs fonctions, chacune dédiée à un objectif particulier. Le programme fera appel aux fonctions au fur et à mesure de son exécution.
- Ecrit sous la forme d'une combinaison de fonctions, le programme sera plus lisible et plus facile à faire évoluer qu'un programme écrit de manière monobloc. De plus, il sera parfois possible de réutiliser certaines fonctions dans d'autres programmes.
- Enfin, la création d'une fonction permet de lutter contre la duplication de code : plutôt que de dupliquer le même code dans un programme, on centralise ce code sous la forme d'une fonction et on y fait appel depuis tous les endroits où c'est nécessaire.

Possibilités des fonctions

- **Valeur de retour**

Voici une variante de notre programme d'exemple.

```
function direBonjour() {
  return "Bonjour !";
}

console.log("Début du programme");
const resultat = direBonjour();
console.log(resultat); // "Bonjour !"
console.log("Fin du programme");
```

Ce programme produit exactement le même résultat que le précédent.

Dans cet exemple, le corps de la fonction **direBonjour**() a été modifié :

l'instruction ***console.log("Bonjour !")*** a été remplacée par la ligne **return** "Bonjour !".

L'utilisation du mot-clé return dans une fonction permet de lui donner une valeur de retour.

Son appel produit un résultat qui correspond à la valeur placée juste après le return dans la fonction.

Ce résultat peut être récupéré par le programme appelant. Ici, la fonction **direBonjour**() renvoie la valeur ***chaîne "Bonjour !"*.**

Cette valeur est stockée par le programme dans la variable resultat, qui est ensuite affichée.

Une fonction incluant une instruction return renvoie une valeur de retour lorsqu'elle est appelée :

cette valeur résulte de l'évaluation de l'expression située immédiatement après le return.

```
// Déclaration d'une fonction nommée maFonction
function maFonction() {
  // Calcul de la valeur de retour
  // ...
  return valeurRetour;
}

// Récupération de la valeur de retour de maFonction
let valeur = maFonction();
// ...
```

Cette valeur de retour peut être de n'importe quel type (nombre, chaîne, etc). En revanche, une fonction ne peut renvoyer qu'une seule valeur.

Rien n'oblige à récupérer la valeur de retour d'une fonction, mais dans ce cas, cette valeur est "oubliée" par le programme qui appelle la fonction !

Si on essaie de récupérer la valeur de retour d'une fonction qui n'inclut pas d'instruction return, on obtient la valeur **JavaScript undefined.**

Une fonction qui ne renvoie pas de valeur est parfois appelée une **procédure**.