appris dans article
https://tympanus.net/codrops/2020/06/17/making-stagger-reveal-animations-for-text/

- preloader

placer script dans head du document HTML dans lequel donner un nom de classe à document.documentElement (Document.documentElement renvoie l'Element qui est l'élément racine du document - par exemple, l'élément <html> pour les documents HTML).

```
<!DOCTYPE html>
<html lang="en" class="no-js">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Typography Motion Effect | Codrops</title>
  <meta name="description" content="Recreation of a letter stagger animation with
  <meta name="keywords" content="stagger, gsap, letter, text, typography, animati
  <meta name="author" content="Codrops" />
  <link rel="shortcut icon" href="favicon.ico">
  <link rel="stylesheet" href="https://use.typekit.net/lwc3axy.css">
  <link rel="stylesheet" type="text/css" href="css/base.css" />
  <script src="https://ajax.googleapis.com/ajax/libs/webfont/1.6.26/webfont.js"><
  <script>
    document.documentElement.className = "js";
    var supportsCssVars = function () {
      var e, t = document.createElement("style");
      return t.innerHTML = "root: { --tmp-var: bold; }", document.head.appendChil
        .supports && window.CSS.supports("font-weight", "var(--tmp-var)")), t.par
    };
    supportsCssVars() || alert("Please view this demo in a modern browser that su
  </script>
</head>

<body class="loading">
```

Cette classe va servir à styliser document le temps du chargement de diverses choses. Ici permet de charger polices de caractères, puis une fois celle-ci chargées dans une promesse className de document.documentElement est enlevée depuis index.js pour laisser place à la page web attendue.

```
src > js > index.js > ...
import "splitting/dist/splitting.css";
import "splitting/dist/splitting-cells.css";
import Splitting from "splitting";
import { gsap } from "gsap";
import { preloadFonts } from "./utils";

preloadFonts("lwc3axy").then(() => document.body.classList.remove("loading"));
```

A noter que le document est stylisé via des pseudo-éléments sans doute pour ne pas "polluer" le DOM.

```css
/* Page Loader */
.js .loading::before,
.js .loading::after {
  content: '';
  position: fixed;
  z-index: 1000;
}

.js .loading::before {
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: lightpink;
  /* background: var(--colo
}

.js .loading::after {
  top: 50%;
  left: 50%;
  width: 60px;
  height: 60px;
  margin: -30px 0 0 -30px;
  border-radius: 50%;
  opacity: 0.4;
  background: red;
```

- découpe minutieuse du html via les classes

```html
<body>
  <main>

    <div class="frame">
      <a class="frame__about">
        <span class="frame__about-item frame__about-item--current">About</span>
        <span class="frame__about-item">Close</span>
      </a>
    </div>

    <div class="content">

      <section class="content__item content__item--home content__item--current">
        <p class="content__paragraph content__paragraph--large content__paragraph--firs
        <p class="content__paragraph content__paragraph--large content__paragraph--righ
          data-splitting>your strength</p>
      </section>

      <section class="content__item content__item--about">
        <p class="content__paragraph" data-splitting>Hello!!!</p>
        <figure class="content__figure">
          <img class="content__figure-img" src="img/venezia.jpeg" alt="Venezia" />
          <figcaption class="content__figure-caption">
            Que c'est beau! Lorem, ipsum dolor sit amet consectetur adipisicing elit. U
            tempora ducimus, vero quos reprehenderit, numquam, in enim dolore doloremqu
            temporel
          </figcaption>
        </figure>
        <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Velit eos asperiore
```

- DOM recréé dans index.js sous forme d'objet pour les besoins de GSAP et autres manipulations du DOM, donne un côté très carré et clean à manipulation du DOM avec Javascript. Semble découper par view (ici "home" et "about"), ainsi que par éléments présents dans chaque view (ici links).

- GSAP semble suivre plus ou moins tout le temps même schéma pour gérer animations.

Timeline composée de tweens, lesquels sont des images générés lors de la transformation d'une image Y vers Z. Par exemple la transformation d'une voiture en vache. Toutes ces

images intermédiaires possèdent à divers degrés les attributs de Y et Z, d'après un dictionnaire des termes d'informatique.

premier tween:

```
const timelineSettings = {
  staggerValue: 0.014,
  charsDuration: 0.5,
};

const timeline = gsap
  .timeline({ paused: true })
  .addLabel("start")
  .staggerTo(
    DOM.content.home.chars,
    timelineSettings.charsDuration,
    {
      ease: "Power3.easeIn",
      y: "-100%",
      opacity: 0,
    },
    timelineSettings.staggerValue,
    "start"
  )
```

1. Ici créé une timeline avec gsap.timeline (A Timeline is a powerful sequencing tool that acts as a container for tweens and other timelines, making it simple to control them as a whole and precisely manage their timing.)
2. .addlabel("start") (Adds a label to the timeline, making it easy to mark important positions/times. You can then reference that label in other methods, like `seek("myLabel")` or `add(myTween, "myLabel")` or `reverse("myLabel")`.)
3. .staggerTo permet de créer une animation dont les éléments à animer le seront de manière décalée. Tweens an array of targets to a common set of destination values, but staggers their start times by a specified amount of time, creating an evenly-spaced sequence with a surprisingly small amount of code.

   Les paramètres sont renseignés comme suit:
   .staggerTo(targets:Array, duration:Number, vars:Object, stagger:Number | Object | Function, position:*, onCompleteAll:Function, onCompleteAllParams:Array, onCompleteScope:*) : *
   a) DOM.content.home.chars correspond aux éléments à animer, ici correspond aux caractères des paragraphes de section "home".
   b) timelineSettings.charsDuration correspond à la durée de l'animation
   c) An object defining the end value for each property that should be tweened as well as any special properties like `ease`.
   d) timelineSettings.staggerValue correspond à délai pour que l'animation commence entre chaque target.

e) label “start” correspond à position. (default = `+=0`) — Controls the placement of the first tween in the timeline (by default, it’s the end of the timeline, like “+=0”). Use a number to indicate an absolute time in terms of seconds (or frames for frames-based timelines), or you can use a string with a “+=” or “-=” prefix to offset the insertion point relative to the END of the timeline. For example, `"+=2"` would place the tween 2 seconds after the end, leaving a 2-second gap. `"-=2"` would create a 2-second overlap. You may also use a label like `"myLabel"` to have the tween inserted exactly at the label or combine a label and a relative offset like `"myLabel+=2"` to insert the tween 2 seconds after “myLabel” or `"myLabel-=3"` to insert it 3 seconds before “myLabel”.

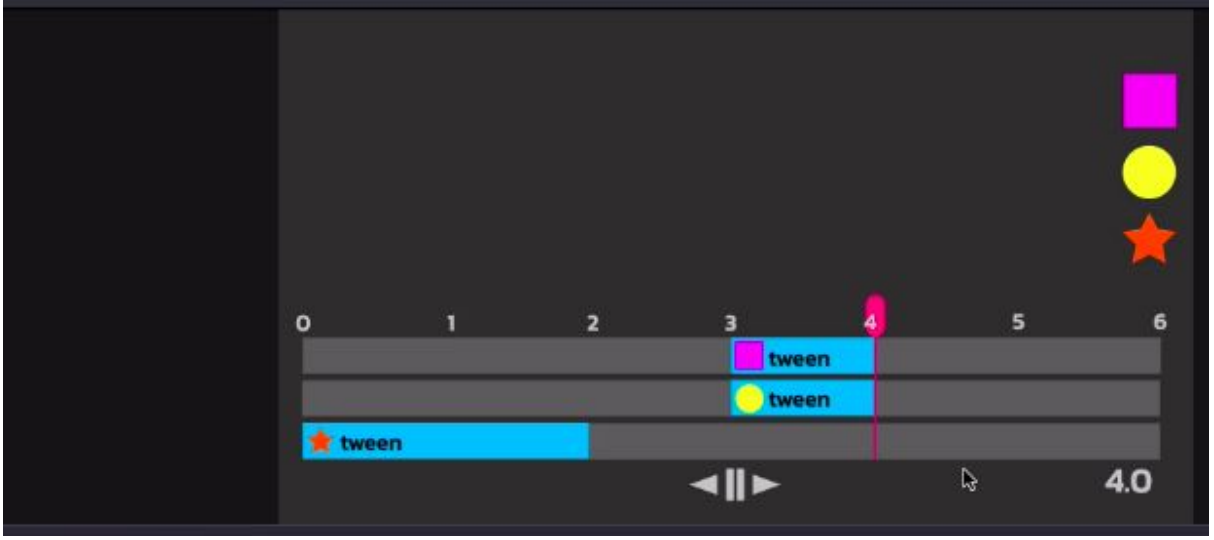
A propose de “position”:
les positions d’animation sont relatifs les uns aux autres ainsi si on utilise valeur par défaut elles se suivront les unes les autres sans délai. En précisant une position on peut les faire se chevaucher. Position relative entre guillemets position absolue sans guillemets.

You can define the position in any of the following ways

- At an absolute time (`1`)
- Relative to the end of a timeline allowing for gaps (`"+=1"`) or overlaps (`"-=1"`)
- At a label (`"someLabel"`)
- Relative to a label (`"someLabel+=1"`)
- Relative to the previously added tween (`"<"` references the most recently-added animation's START time while `">"` references the most recently-added animation's END time)

```
var animation = gsap.timeline();
animation
      .to("#star", {duration:2, x:1150})
      .to("#circle", {duration:1, x:1150}, "+=1")
      .to("#square", {duration:1, x:1150})
```


```
var animation = gsap.timeline();
animation
      .to("#star", {duration:2, x:1150})
      .to("#circle", {duration:1, x:1150}, "+=1")
      .to("#square", {duration:1, x:1150}, "-=1")
```



second tween:

src > js > index.js > timeline

```
65    .addLabel("switchtime")
66    .add(() => {
67      DOM.content.home.section.classList.toggle("content__item--current");
68      DOM.content.about.section.classList.toggle("content__item--current");
69    })
70    .to(
71      document.body,
72      {
73        duration: 0.8,
74        ease: "Power1.easeInOut",
75        backgroundColor: "#c3b996",
76      },
77      "switchime-=timelineSettings.charsDuration/4"
78    )
79    .set(
80      DOM.content.about.chars,
81      {
82        y: "100%",
83      },
84      "switchtime"
85    )
86    .set(
87      DOM.content.about.picture,
88      {
89        y: "40%",
90        rotation: -4,
91        opacity: 0,
92      },
93      "switchime"
94    )
95    .staggerTo(
96      DOM.content.about.chars,
97      timelineSettings.charsDuration,
98      {
99        ease: "Power3.easeOut",
100       y: "0%",
101     },
102     timelineSettings.staggerValue,
103     "switchtime"
104   )
105   .to(
106     DOM.content.about.picture,
107     0.8,
108     {
109       ease: "Power3.easeOut",
110       y: "0%",
111       opacity: 1,
112       rotation: 0,
113     },
114     "switchtime+=0.6"
```

1. .addLabel(“switchtime”) marque un nouveau moment important dans la timeline auquel on pourra se référer pour gérer position des tweens suivants.
2. .add(cb), ici modifie simplement classList des sections au début du tween en cours.
   .add(value:*, position:*, align:String, stagger:Number) : *

[override] Adds a tween, timeline, callback, or label (or an array of them) to the timeline.

3. .to est The most common type of animation is a to() tween because it allows you to define the destination values, dont les paramètres sont:

a) targets - the object(s) whose properties you want to animate. This can be selector text like `".class"`, `"#id"`, etc. (GSAP uses `document.querySelectorAll()` internally) or it can be direct references to elements, generic objects, or even an array of objects.
b) vars - an object containing all the properties/values you want to animate, along with any special properties like `ease`, `duration`, `delay`, or `onComplete`.
c) position

4. .set est Immediately sets properties of the target(s) accordingly - essentially a zero-duration to() tween with a more intuitive name. So the following lines produce identical results:

```
gsap.set(".class", {x: 100, y: 50, opacity: 0});
gsap.to(".class", {duration: 0, x: 100, y: 50, opacity: 0});
```

Ici en position utilise label “switchtime” donc au début de tween “switchtime” est appliqué

5. idem
6. .staggetTo() et .to() modifient valeurs set juste avant avec .set(), l’intérêt peut-être de enfin, créer ce mouvement de déplacements des éléments (ci-étaient à leur position initiale dès le changement de page il ne pourrait y avoir d’animation de transition il me semble), de plus je pense que ça sert à replacer les éléments durant la navigation pour que l’animation soit jouée à chaque fois. .to() pour l’image est placée en position “switchtime+=0.6” soit 0.6 secondes après le début de tween “switchtime” pour créer cette effet de décalage entre l’affichage des éléments.

- Enfin, fonction créée laquelle ajoute ou enlève class current selon que est visible ou pas d’après propriété “isVisible”. Toujours selon propriété “isVisible’ gsap timeline est play() ou reverse(), puis sont inversées les valeurs de “isVisible” des différentes sections.

```
117 const switchContent = () => {
118   DOM.links.about.stateElement[0].classList[DOM.content.about.isVisible ? "add" : "remove"]("content__item--current");
119   DOM.links.about.stateElement[1].classList[DOM.content.about.isVisible ? "remove" : "add"]("content__item--current");
120   timeline[DOM.content.about.isVisible ? "reverse" : "play"]();
121   DOM.content.about.isVisible = !DOM.content.about.isVisible;
122   DOM.content.home.isVisible = !DOM.content.home.isVisible;
123 };
124
125 DOM.links.about.anchor.addEventListener("click", () => switchContent());
126
```