```
button {
  padding: 1rem 2rem;
  font-size: 1.5rem;
  border: none;
  color: white;
  background: #404040;

  transition: background 0.5s ease-out, box-shadow 0.5s ease-in;

  box-shadow: 2px 2px blue;
}

button:hover {
  background: #707070;
  box-shadow: 1px 1px lightgrey;
}
```

****

From Here We Must Add The Transition To The Main Element Not The Class:

```
nav {
  display: flex;
  justify-content: center;
  align-items: center;
  background: lightgrey;
  height: 10vh;
  color: lightslategray;
  transition: transform 1s ease-out, height 1s ease-out;
}
```

```
.nav-slide {
  transform: translateY(-20vh); /* The Wrong Way */
  height: 0;
}
```

****

Note (From Me): The Correct Way To Animate The List by using height and opacity without transform.

```
nav {
  display: flex;
  justify-content: center;
  align-items: center;
  background: lightgrey;
  height: 10vh;
  color: lightslategray;
  transition: all 0.5s ease-out;
  -webkit-transition: all 0.5 ease-out;
  -moz-transition: all 0.5s ease-out;
  -o-transition: all 0.5s ease-out;
}

.nav-slide {
  height: 0;
  opacity: 0;
}
```

****

This Will Make The Text Disappear Then Slide Transition

```
nav {
  display: flex;
  justify-content: center;
  align-items: center;
```

```
 background: lightgrey;
 height: 10vh;
 color: lightslategray;
 transition: all 0.5s ease-out 0.5s;
}


p {
 opacity: 1;
 transition: opacity 1s ease-out;
}

.fade {
 opacity: 0;
}
```
****

This Will Make The Text Appear After The Slide is In Place.

```
p {
 opacity: 1;

 transition: opacity 1s ease-out 0.5s;
}
.fade-02 {
 opacity: 0;
}
```
****

To Animate SVG Images Using JS:

- Make Anu Changes In Any UI/UX Tools Like Figma.
- Group The Content That We Want.
- Export The SVG Image With Ids.
- Set The SVG Image AS Element In Our Project.

- Set The Class Name For The SVG-Element.
    - <svg class="cookie" width="98" ="98" viewBox="0 0 98 98" fill="none" xmlns=http://www.w3.org/2000/svg>

****

To Make Container In Center Of Screen With position: absolute we can use transform with translate:

```
.cookie-container {
 background: linear-gradient(260deg, #9b6c50 0%, #4f2626 100%);
 position: absolute;
 top: 50%; // important
 left: 50%; // important
 padding: 10px;
 transform: translate(-50%, -50%);
}
```

****

This GSAP Code Will Make The Result is The End Show:

```
gsap.to('.text', { y: 140, opacity: 0, duration: 3 });
```

****

This GSAP Code Will Start using From-options And Finish In To-option:

```
gsap.fromTo('.text', { opacity: 0, y: 20 }, { opacity: 1, y: 0, duration: 3 });
```

The Duration Must Be In To-Option.

****

TimeLines In GSAP can make us combine multiple Animations (To) Inside One Command.

TimeLines Combine: The Duration, The Ease, The Defaults Values, ...etc.

****

```
const tl = gsap.timeline({
      defaults: {
         duration: 1
      }
   });
tl.fromTo('.cookie-container', { scale: 0 }, { scale: 1});
**********
```

In This Way, We Can Chain The Animations Together:

```
const tl = gsap.timeline({
   defaults: {
    duration: 1,
   },
});
tl.fromTo(".cookie-container", { scale: 0 }, { scale: 1 });
tl.fromTo(".cookie", { opacity: 0, x: -50 }, { opacity: 1, x: 0 });
**********
```

In This Way We Can Sync The Animation, By Adding < To The Animation, It Will Be Sync it With The Previous One:

```
const tl = gsap.timeline({
      defaults: {
      duration: 1,
      },
   });
tl.fromTo(".cookie-container", { scale: 0 }, { scale: 1 });
tl.fromTo(
            ".cookie",
      { opacity: 0, x: -50, rotation: "-45deg" },
      { opacity: 1, x: 0, rotation: "0deg" }
);
tl.fromTo(".text", { x: 30, opacity: 0 }, { x: 0, opacity: 1 }, '<');
**********
```

In This Way By Adding <25%, This Will Sync The Animation All Together, But After The 25% Of The First One Start:

```
const tl = gsap.timeline({
    defaults: {
    duration: 1,
    ease: 'power1.inOut'
    },
  });

tl.fromTo('.cookie-container', { scale: 0 }, { scale: 1, ease: "elastic.out(1.5,0.5)", duration: 2.5 });

tl.fromTo(
    '.cookie',
    { opacity: 0, x: -50, rotation: '-45deg' },
    { opacity: 1, x: 0, rotation: '0deg' }, '<25%'
);
tl.fromTo('.text', { x: 30, opacity: 0 }, { x: 0, opacity: 1 }, '<');
```

****

In This Way We Can Repeat The Animation, by using:

- Attribute 1 ➔ yoyo: true →Then It Will Repeat The Animation.
- Attribute 2 ➔ repeat: number → This Will Repeat It By #number Of Times
    - If We Set repeat: -1 →This Will Make It Repeat For Ever.

****

If We Set Here The Rotation It Will Override The CSS Properties:

```
tl.fromTo('#crumbs', { y: 0, }, { y: -25, yoyo: true, repeat: -1 }, '<');
```

And To Stop The TimeLine Animation:

```
button.addEventListener('click', ()=> {
    gsap.to('.cookie-container', { opacity: 0, y: 100, duration: 1, ease: 'power1.inOut' });
    tl.pause();
  });
```

****

Note (To Remember): To Make The Animation Start After Specific Percent Of Parent Animation:

Here We Start The Child Animation After 25% Of Parent Start Animation:

```
tl.fromTo('.cookie-container', { scale: 0 }, { scale: 1, ease: "elastic.out(1.5,0.5)", duration: 2.5 });
tl.fromTo(
    '.cookie',
    { opacity: 0, x: -50, rotation: '-45deg' },
    { opacity: 1, x: 0, rotation: '0deg' }, '<25%'
  );
```

****

When Image Doesn't Respect The Height And Width Of Parent We Must Set Its Height And Width to: 100%

```
.hero-section {
  height: 80vh;
  margin: 0% 10%;
}

.hero-section img {
  width: 100%;
  height: 100%;
```

```
}
```

****

To Make Image Respect Its Size When Change The Browser Window Size, We Can Set: object-fit: cover;

```
.hero-section img {
  width: 100%;
  height: 100%;
  object-fit: cover;
}
```

****

The span-Element Doesn't Respect The X, Y Of GSAP, So We Must Change The Display Of It.

```
.cta-text span {
  padding: 0 0.5rem;

  text-shadow: rgba(0, 0, 0, 0.5) 0px 0px 5px;

  display: block;
}
```

```
tl.fromTo('.cta-01', { x: 110, opacity: 0 }, { x: 0, opacity: 1 });
```

****

The Best Way To Make The Text Disappear first, by make x to 100%:

```
tl.fromTo('.cta-01', { x: "100%", opacity: 0 }, { x: 0, opacity: 1 });
```

****

To Update The Elements One By One That Has The Same Class, We Can Use Stagger:

```
const logo = document.querySelector('.logo');
console.log(logo);

const letters = logo.textContent.split('');
console.log(letters);

logo.textContent = '';

letters.forEach(letter => {
    logo.innerHTML += `<span class="letter">${letter}</span>`;
});
gsap.set('.letter', { display: 'inline-block' });
gsap.fromTo('.letter', { opacity: 0, y: -30 }, { opacity: 1, y: 0, stagger: 0.075, delay: 2.5 });
```

****

Note (To Remember): Always To Hide The Elements That Are Outside Of Their Container Use: ***overflow:hidden***

****

Note (To Remember): When Handling Flex, be attention to align-items, and justify content, else The Content Will be Not Alignment Correctly.

****

When We Want Our Animation to Return To Previous State When using ***.fromTo*** We Can Add ***yoyo:true***

****

To Make The SVG Transformed From Center:

```
gsap.set('.feather', { scale: 0, transformOrigin: 'center' });
```

****

If We Want To Define The Stagger Duration Between The Components We Can Set: duration to stagger like: stagger: 0.5 that means 500ms.

```
tl.fromTo('.feather', { y: -5, scale: 0}, { y: 20, scale: 1.5, duration: 2.5, stagger: 0.5  }, '<50%');
```

****

If We Have Different View Of Animation Using timeline Then Try Use gsap-instead.

```
home.addEventListener('click', () => {
    // console.log('Home Clicked');
    gsap.fromTo('.home-svg', { scale: 1 }, { scale: 0.75, yoyo: true, repeat: 1, ease: "expo.out", duration: 1 });
    gsap.fromTo('.feather', { y: -5, scale: 0}, { y: 20, scale: 1.5, duration: 2.5, stagger: 0.5  });
    gsap.fromTo('.right-feather', { x: 0 }, { x: 5 });

  });
```

****

To Change The Origin Of Rotation, Scaling, ...etc: We Can Change The transformOrigin:

```
gsap.set('.bell', { transformOrigin: 'top center' });

gsap.set('.ringer', { transformOrigin: 'bottom center' });

gsap.set('.wave', { opacity: 0, transformOrigin: 'bottom' });
```

****

```
gsap.fromTo('.bell', { rotation: -20 },
      { rotation: 0, duration: 2.5, ease: "elastic.out(2,0.2)", }
);
gsap.fromTo('.ringer',
      { rotation: -20, x: 1 },
      { rotation: 0, x:0, duration: 2.5, ease: "elastic.out(2,0.2)", }
);
gsap.fromTo('.wave', { opacity: 1, scale: 0 }, { opacity: 0, scale: 1.2, duration: 2.5});
```

****

To Reverse The Scaling Operation We Can Set:

```
gsap.set('.flap', { transformOrigin: 'top'});

tl.fromTo('.flap', { scale: 1 }, { scale: -1 });
```

****

To Make Our Animation Make In Same Timeline We Can Use:

```
messages.addEventListener('click', () => {
// console.log('messages Clicked');
      tl.fromTo('.messages-svg', { scale: 1 }, { scale: 0.9 });

      tl.fromTo('.flap', { scale: 1 }, { scale: -1 }, '<25%');

      tl.fromTo('.messages-svg', { scale: 0.9 }, { scale: 1 }, '<75%');

      tl.fromTo('.note', { opacity: 1, y: 0 }, { opacity: 0, y: -50,
duration: 1.5 }, '<25%');

      tl.to('.flap', { scale: 1 }, '<75%');
});
```

****