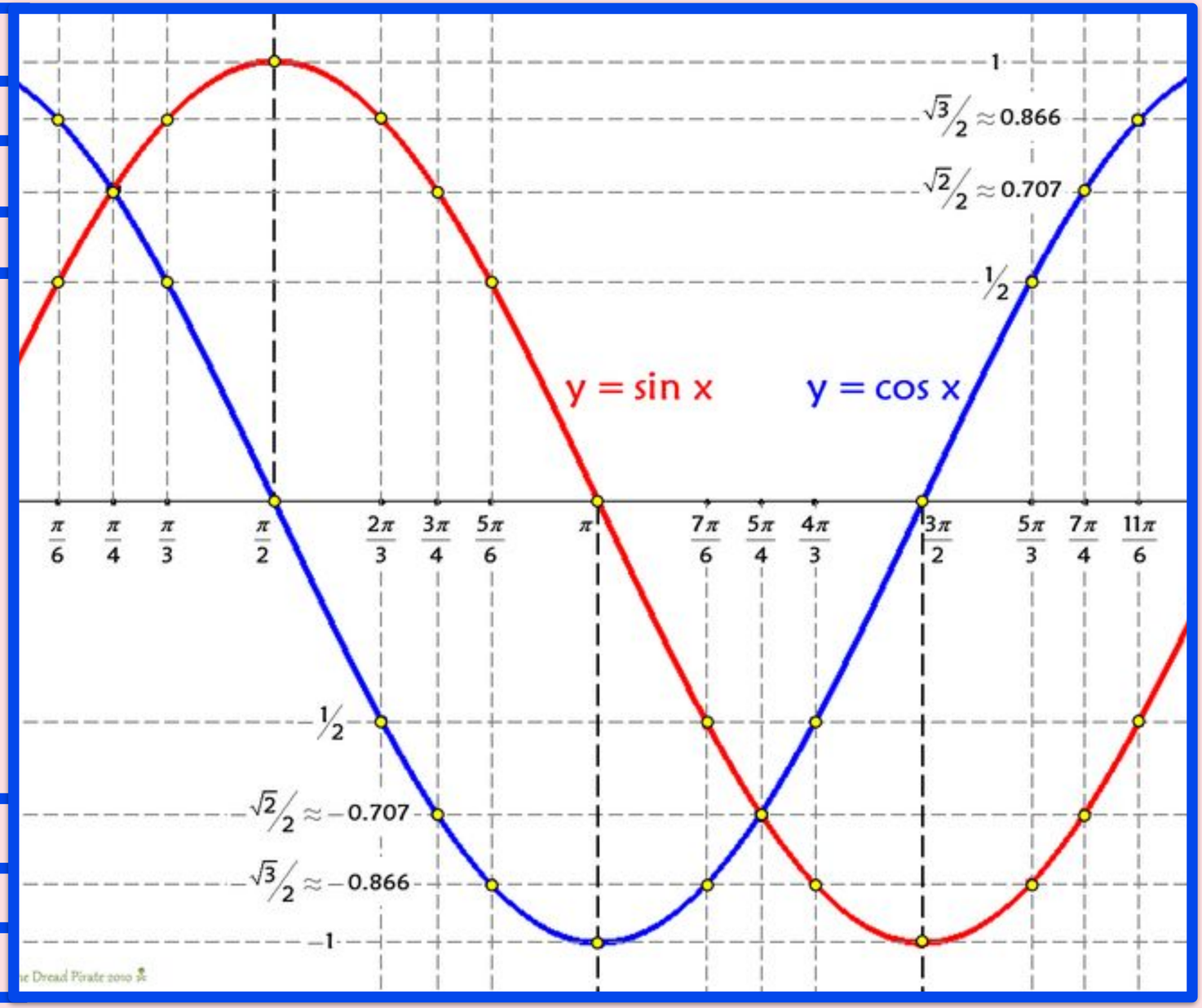
Week 4

PIXI: Animation

Sinusoidal

Motion



Sine of Time

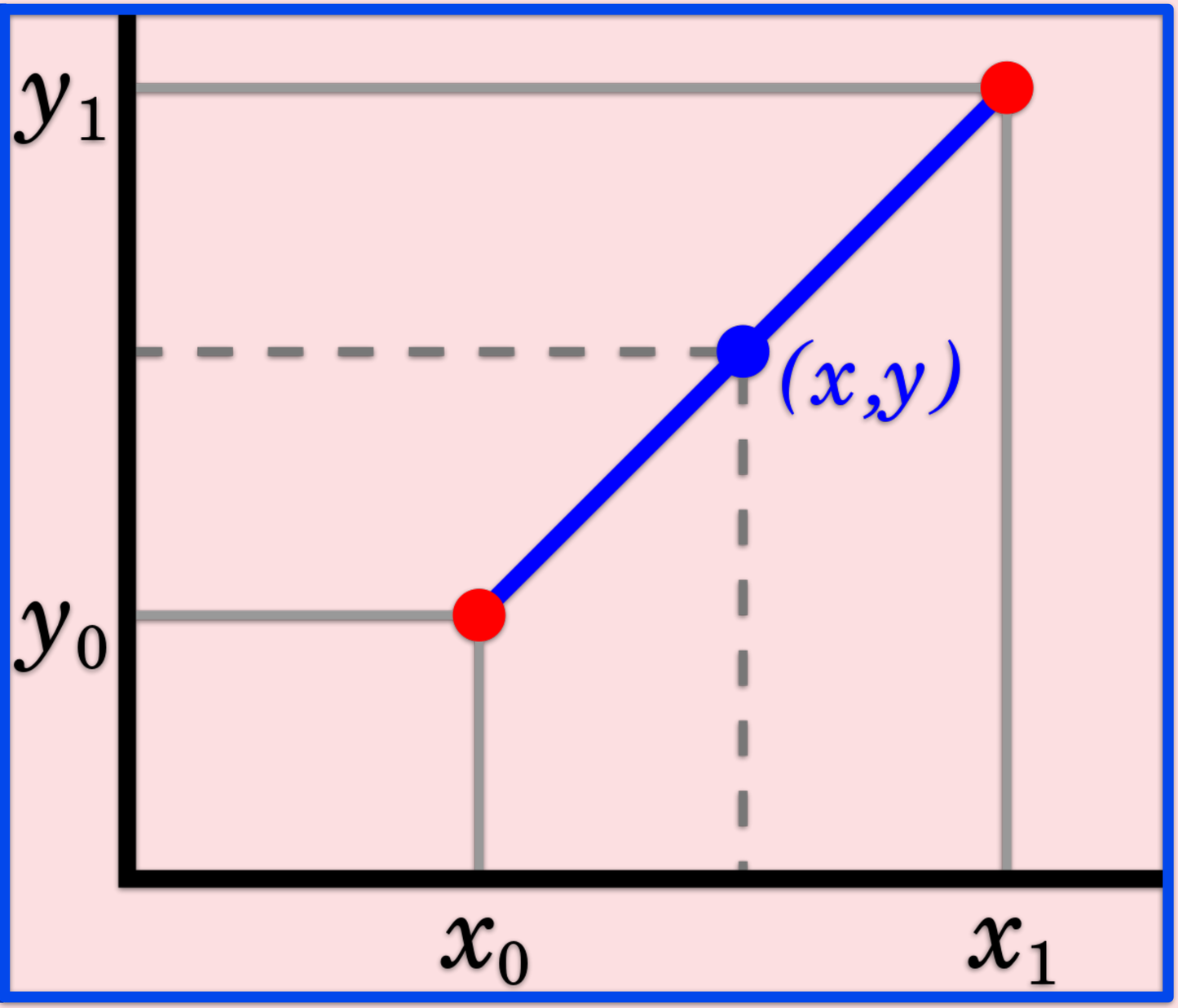
A tried and true way to animate things cyclically is to use Sine and/or Cosine functions of time

```
// move something in a circular motion

const radius = 100
mySprite.x = Math.cos(time) * radius
mySprite.y = Math.sin(time) * radius
```

Linear

Interpolation



lerp

My using linear interpolation, we can modify a property by a given percentage between two values

```
function lerp(a:number, b:number, pct:number): number {
    return a * (1-pct) + b * pct;
}
```

easeIn

My using linear interpolation, we can modify a property by a given percentage between two values

```
function easeIn(t: number): number {
     return t * t;
}

let value = lerp(x, y, easeIn(time));
```

easeOut

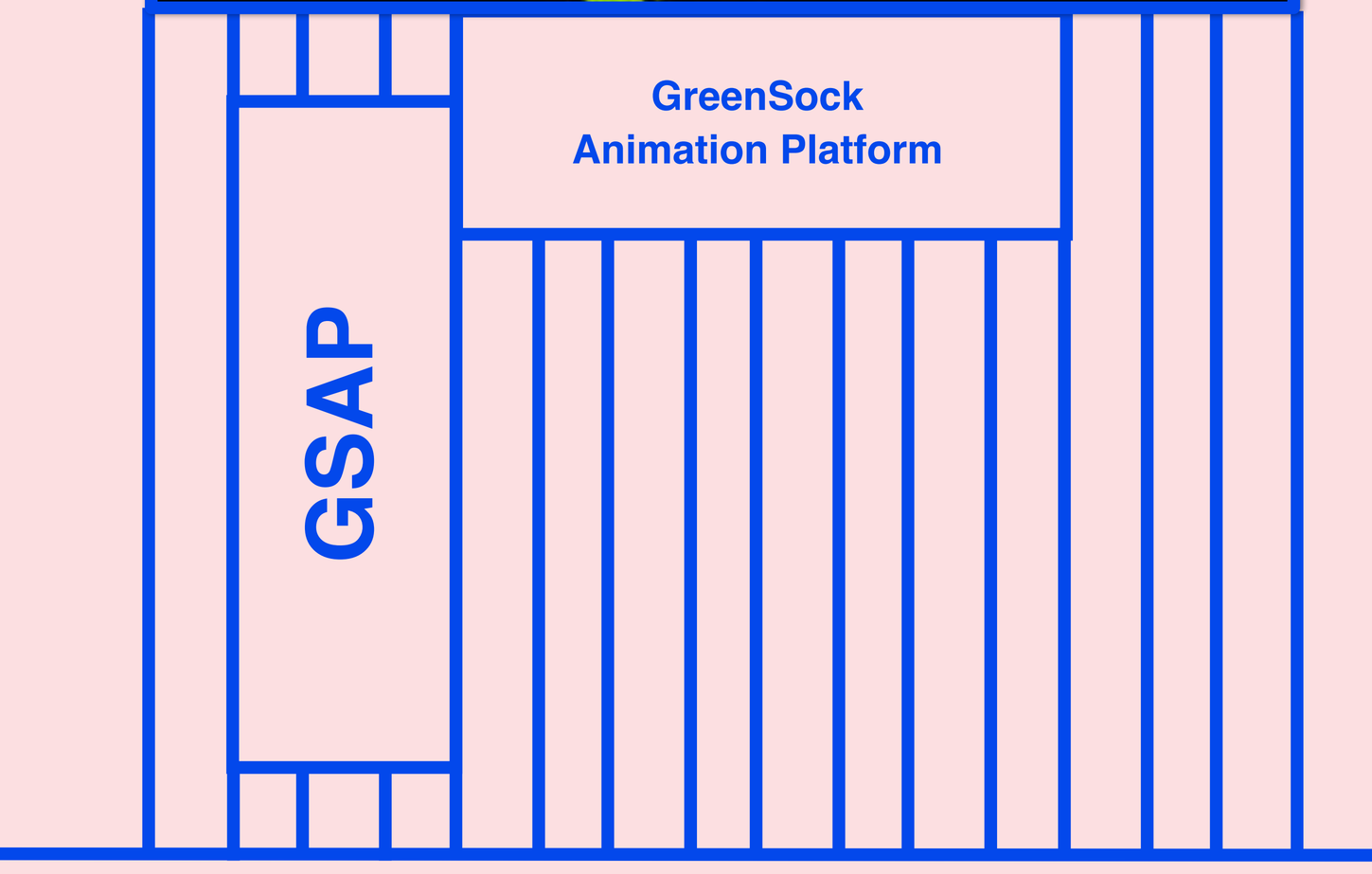
My using linear interpolation, we can modify a property by a given percentage between two values

```
function flip(x: number): number {
  return 1 - x;
}

function easeOut(t: number): number {
  return flip(easeIn(flip(t)))
}

let value = lerp(x, y, easeOut(time));
```


GSAP
GreenSock Animation


GreenSock
Animation Platform
GSAP

Tweens

Gsap lets us be more declarative about the easing that we apply to objects in our scene

```
> npm install --save gasp

var obj = {prop: 10};

gsap.to(obj, {
  duration: 1, // time in seconds of the animation
  prop: 200, // property to change

  onUpdate: function() { // called every update
    console.log(obj.prop);
  }
});
```

Easing and Timescale

We can also assign the shape of the easing of the animation to our objects, as well as the timescale (or playback rate)

```
let obj = {prop: 10};

gsap.to(obj, {
  duration: 1, // time in seconds of the animation
  prop: 200, // property to change
  ease: “pow2”
  onUpdate: function() { // called every update
    console.log(obj.prop);
  }
});
```

Tween Methods

We can also add commands for playing/ pausing, resetting, reversing and stopping our tweening

```
tween.pause();

tween.resume();

tween.reverse();

tween.seek(0.5);

tween.timeScale(0.5);
```

Chaining Tweens

A tween can be treated as a promise, so after we've defined a tween, we can use await or .then() on it

```
let tween = gsap.to(object,{...})


tween.then(() => {
  // some callback instructions...
})
```

Timelines

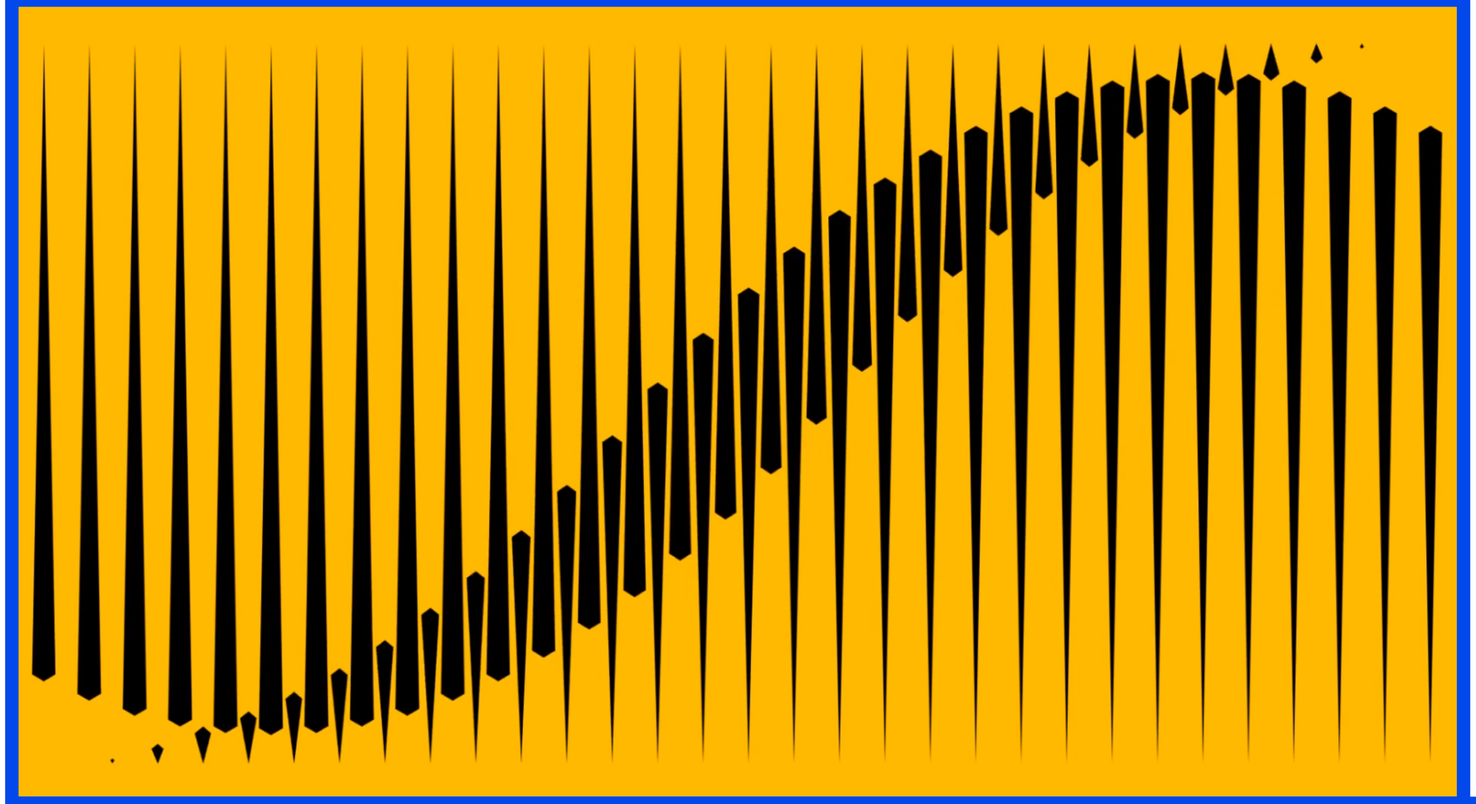
We can use timelines to compose tweens together and/or stagger animations among many objects in our scene

```
tl.to(element, {x: 200})

  .to(element, {y: 200}, "+=1") //1 second after end of timeline (gap)

  .to(element, {rotation:90}, "-=0.5") //0.5 seconds before end of timeline (overlap)

  .to(element, {scale: 4}, 6); //at exactly 6 seconds from the beginning of the timeline (absolute)
```

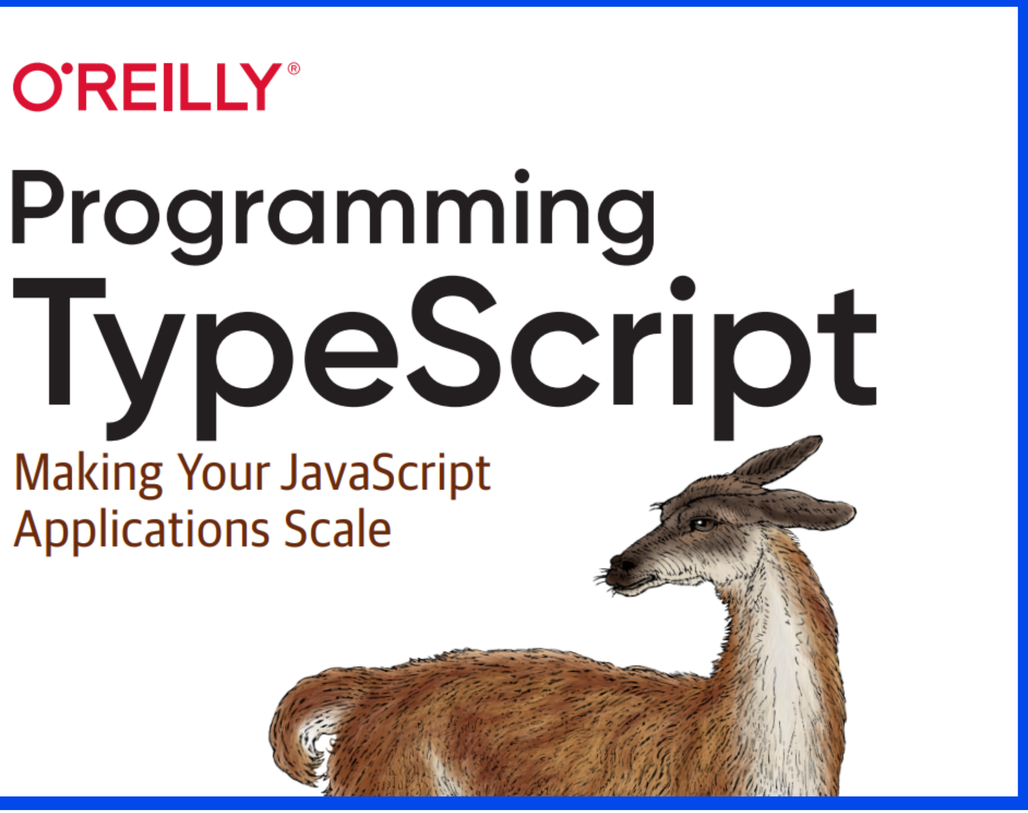


Homework for next week

Waves

Watch this music video and use what you've learned this week to try reproducing scenes or shots from this in Pixijs

https://vimeo.com/178612704



Homework for next week

Readings:

What Screens Want, Frank Chimero

Optional readings (Links on Brightspace):

- gsap docs

- https://www.youtube.com/playlist?list=PLugegG07di3886WYN6u7v9BeBd0VFG3_J

- https://www.youtube.com/watch?v=PKZJmHrG4Yw

- https://www.febucci.com/2018/08/easing-functions/

Model

We can make ourselves a class which has all the global data that we need to inform our app. Most models conform to the Singleton pattern, which makes sure that there’s only one of in in our app

```
class Model{
  private static instance: Model;

  private constructor(){
    if(Model.instance) {
      Return Model.instance
    }
    Model.instance = this
  }

  public static get Instance()
  {
    return this._instance || (this._instance = new this());
  }
}
```

View

In Pixi, we can make a class that acts as a Container for it’s child objects

```
Class BaseScene {
  private model: Model;
  public container: Container;
  constructor(model:Model) {…}
  update() {…}
}
```