this

JS Logic

Evaluations. Analysing values in scripts to determine whether or not they match expected results.

Decisions. Using results of evaluations deciding which path script should take.

Loops. When we want to perform the same set of steps repeatedly.

```
if (theTemp < 32) {
    document.write(“boo - it’s a snow day class is cancelled.”)
    } else {
    document.write(“Yay! We can haz class!”)
}
```

Conditional Statements

There are two components to a decision:
An expression is evaluated, which returns a value (checking the current status and returning true or false)

A conditional statement says what to do in a given situation (which path to take)

JS Logic

Comparison operators

== (is equal to)

Compares two values to see if they are the same

!= (is not equal to)

Compares two values to see if they are not the same

=== (strict equal to)

Compares two values to check that both the data and value are the same

!== (strict not equal to)

Compares two values to check that both the data and value are not the same

>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

Enclosing parentheses Comparison operator

(temperature < 32)

Operand Operand

false

Expression 3

((temperature < 32) && (month == “January”))

Expression 1 **true** Expression 2 **false**

Logical and
&&

Tests more than one condition
((temperature < 32) && (month == “January”))

If both expressions return true, then the full expression returns true

true && true returns **true**
true && false returns **false**
false && true returns **false**
false && false return **false**

Logical or

||

Tests at least one condition

((temperature < 32) || (month == “January”))

If either expression return true, then the full expression returns true

Logical not

!

Takes a single boolean value + inverts it

! (temperature < 32)

! true returns **false**

! false returns **true**

When do we need JavaScript Time

+ When we want to know what today (or any other day) is
+ When we want to execute something at a certain time
+ When we want to execute something repeatedly

For this we can use JavaScript Date object that has a lot of different built in methods and parameters.

All date functionality references January 1st, 1970, known as **Unix time**.

Methods you can use

+ **Date( );** - get current date
+ Date.now( ); - the number of milliseconds since January 1, 1970
+ var today = new Date(Date( )); - creates a new date object in a variable
+ today.getDate( ); - returns the day of the month (1-31)
+ today.getDay( ); - returns the day of the week (0-6, where Sunday - 0)
+ today.getFullYear( ); - returns the year
+ today.getHours ( ); - returns the hour (0-23)
+ today.getMillliseconds( ); - returns the milliseconds (0-999)
+ today.getMonth( ); - returns the month (0-10)
+ today.getSeconds( ); - returns the seconds (0-59)
+ today.getTime( ); - returns the number of milliseconds since midnight January 1, 1970, and a specified date

How to count elapsed time

```
// Get current time in milliseconds
        var theStart = Date.now( );

// Something happens for a long time…

// Get new current time in milliseconds
      var end = Date.now();
      var elapsed = end - start; // elapsed time in milliseconds
```

setTimeout, setInterval

We use setTimeout to execute function at a certain time (e.g. in 2 seconds)
We use setInterval when we want something to happen repeatedly

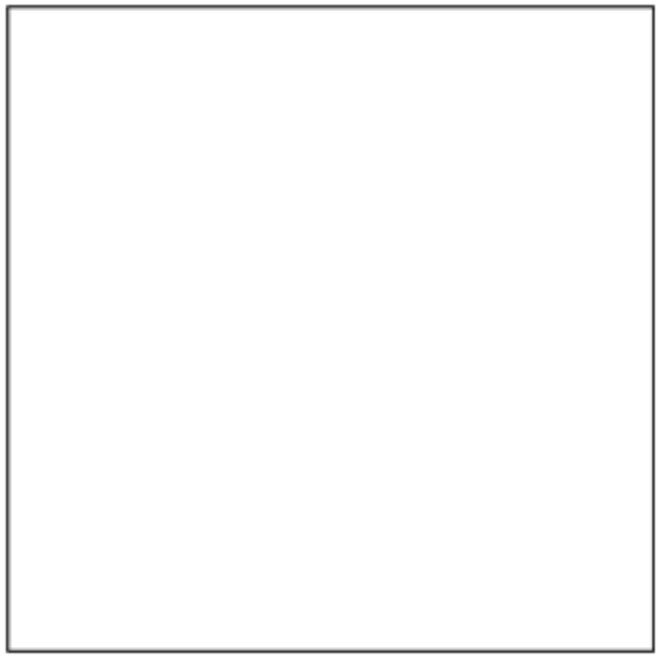
```
function alertMe( ) {
        alert(“It’s me!”);
}
setTimeout(alertMe, 2000);

function alertMe( ) {
        alert(“It’s me!”);
}

setInterval(alertMe, 2000);
```

Clearing timeout or interval

We assign setTimeout, setInterval to a variable
And use clearTimeout or clearInterval

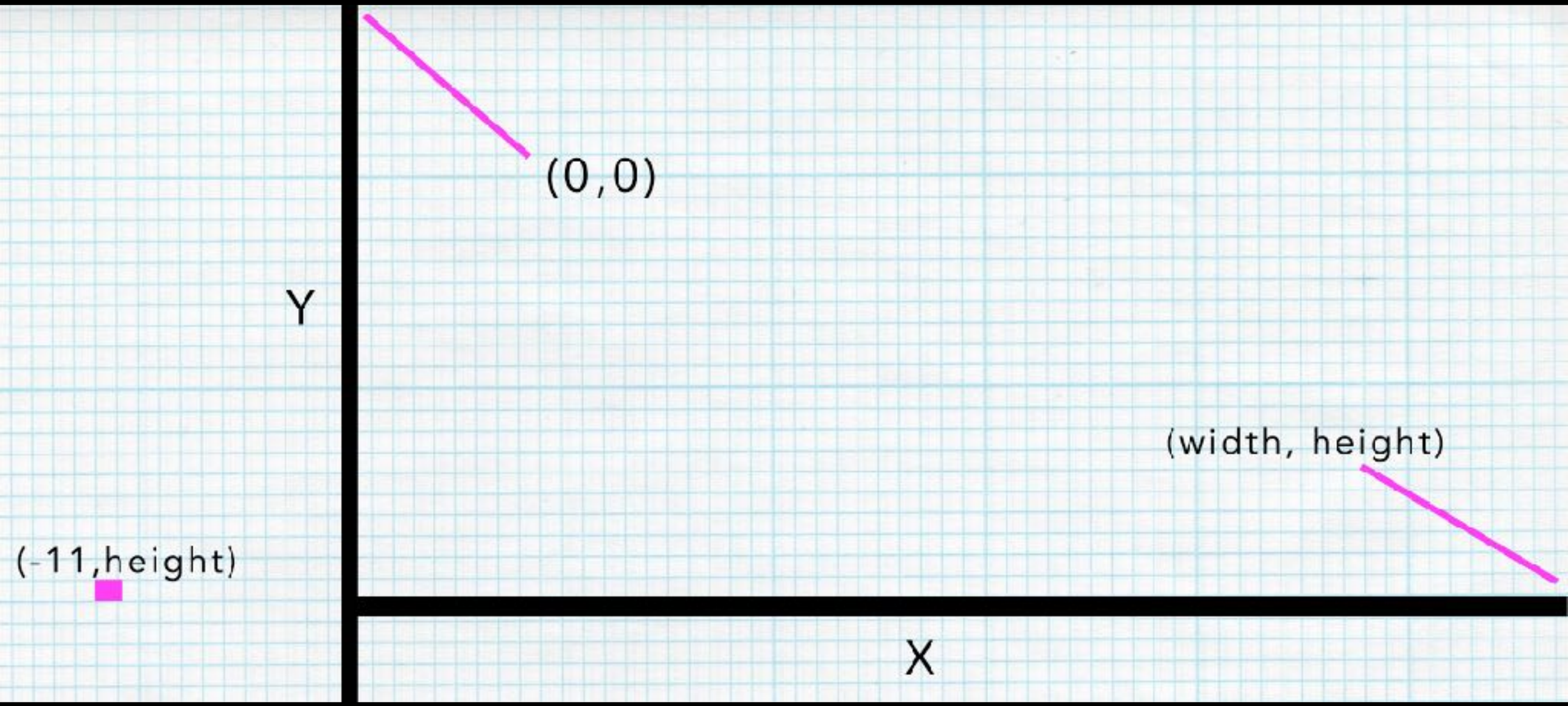
```
function alertMe( ) {
        alert(“It’s me!”);
}
var alertIs = setInterval ( alertMe, 2000);

function stopAlert( ) {
        clearInterval ( alertIs );
}

setTimeout(stopAlert, 10000); // will stop after 5 alerting 5 times
```

<canvas> is used to draw graphics on the webpage



```
<canvas id="myCanvas" width="300px" height="300px" style="border: 1px solid black"></canvas>
```

the canvas

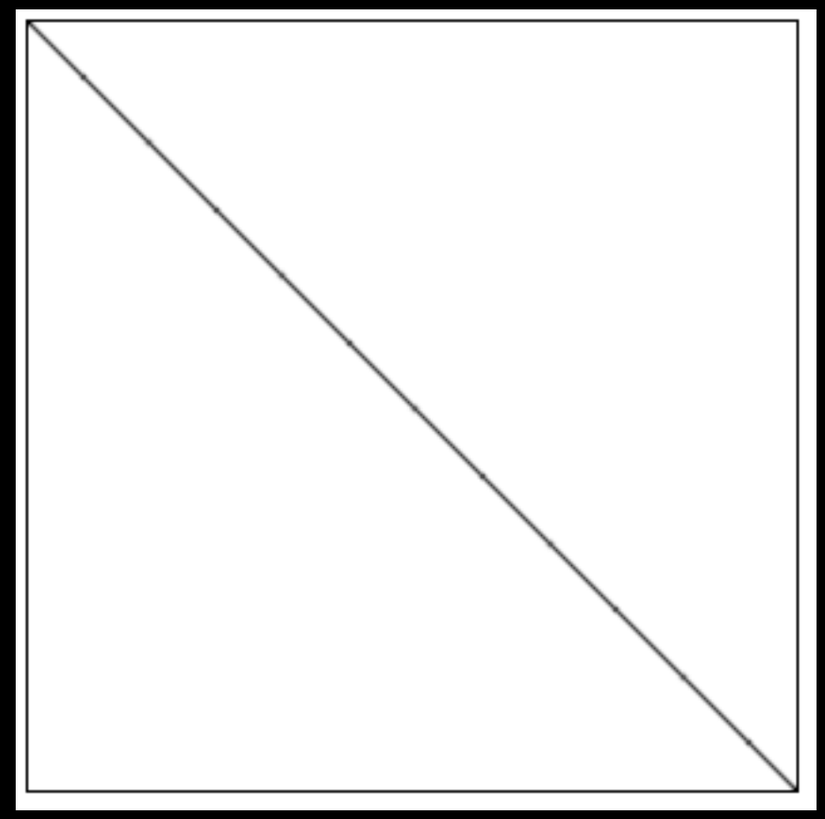


Canvas

+ Always define width and height
+ Give id (because you'll have to call it in JavaScript)
+ Canvas coordinates start from 0

Drawing a line

```
var theCanvas = document.getElementById("myCanvas");
var theContext = theCanvas.getContext("2d");
theContext.moveTo(0,0);
theContext.lineTo(300,300);
theContext.stroke( );
```



Built-in Canvas Methods

+ rect( ) - creates a rectangle
+ stroke( ) - draws a path we have defined
+ beginPath( ) - begins a path
+ arc( ) - creates an arc / curve (used to create circles)
+ scale( ) - scale a current drawing bigger or smaller
+ drawImage( ) - draws an image, canvas or video onto canvas
+ others…

Objects group together a set of variables and functions to create a model of something you would recognize from the real world.

Object properties + methods

In an object:

1. **Variables** become known as **properties**
2. **Functions** become known as **methods**

Properties tell us about the object, such as the height of a chair and how many chairs there are -

```
function Chair( height, number ){
    this.h = height;
    this.count = number;

}
```

Methods represent tasks that are associated with the object, e.g. count the height of all chairs by adding all heights together

Properties & objects

```
var hotel = {
  // the following are key: value pairs.
    name: “Radisson”,
    rooms: 55,
    booked: 15,
    spa: false,
    roomTypes: [“single”, “twin”, “suite”],

    checkAvailability: function( ) {
      return this.rooms - this.booked;
    }
}
```

Properties

Method

name, rooms, booked - are **keys** in the object
Object cannot have two keys with the same name
Value can be anything you want (string, number, boolean, function, another object)

Creating an object: literal notation

```
var hotel = {
  // the following are key: value pairs.
    name: "Radisson",
    rooms: 55,
    booked: 15,
    spa: false,
    roomTypes: ["single", "twin", "suite"],

    checkAvailability: function( ) {
      return this.rooms - this.booked;
    }
}
```

Object is stored in a variable called hotel.

Each **key** is separated from its **value** by a colon :

Each property and method is separated by a comma, this keyword indicates that "rooms" and "booked" properties should be taken from this object

Accessing an object

There are two ways to access properties in an object 1) dot notation 2) square brackets

```
var hotel = {
  // the following are key: value pairs.
    name: "Radisson",
    rooms: 55,
    booked: 15,
    spa: false,
    roomTypes: ["single", "twin", "suite"],
    checkAvailability: function( ) {
      return this.rooms - this.booked;
    }
}
```

Properties

Method

dot notation

```
  var hotelName = hotel.name;
  var roomsAvailable = hotel.checkAvailabilty( );
```

square brackets

```
  var hotelName = hotel["name"];
  var roomsAvailable = hotel["checkAvailability"]( );
```

Creating an object: constructor notation

```
var hotel = {
  // the following are key: value pairs.
    name: "Radisson",
    rooms: 55,
    booked: 15,
    spa: false,
    roomTypes: ["single", "twin", "suite"],

    checkAvailability: function( ) {
      return this.rooms - this.booked;
    }
}
```

The new keyword creates a blank object
Then you can add properties and method using dot notation

Updating an object

```
var hotel = {
  // the following are key: value pairs.
    name: “Radisson”,
}

dot notation
    hotel.name = “Hilton”;

square brackets
    hotel[“name”] = “Hilton”;
```

Deleting property from an object

```
var hotel = {
  // the following are key: value pairs.
    name: “Radisson”,

}
```

dot notation

```
delete hotel.name;
```

Creating many objects: constructor notation

Often - you will want several objects to represent similar thing
You can use function as a template to create several objects
First we need to create template with object's properties and methods

```
var theHotel (theName, theRooms, theBooked) = {
  // the following are key: value pairs.
    this.name: theName,
    this.rooms: theRooms,
    this.booked: theBooked,

    checkAvailability: function( ) {
      return this.rooms - this.booked;
    }
}

var radissonHotel = new theHotel("Radisson", 55, 15);
var hiltonHotel = new theHotel("Hilton", 60, 60);
```

Properties

Method

Final Project

Your Final Must be a **CONTEMPORARY** Web Page. It can be a portfolio site or net art piece. I am not setting a page # requirement, but will be looking for evidence that you put effort into making the most technically + aesthetically sophisticated project you can.

If programming seems daunting - your JS can be minimal , but in this case perhaps you will want to become an expert with CSS animations or the like? I will also accept web-based code projects using p5.js - but individual sketches must be linked from a home index page.

Original Content - no memes!
This should be a portfolio piece you can refer to after you graduate to showcase both your design + technical abilities (image-making is also a skill).

HTML structure with CSS styling
CSS - use of classes and IDs, as well as pseudo classes
CSS Positioning and Layout (responsive, bootstrap grid)
JavaScript Interactivity
Thoughtful + Appealing Design
Posted to Your FM Server

Final Project Time-Frame:

Week 13:

Wednesday December 5

* Final Concept Ready
* **In-class final project proposal presentations**

Week 14:

Wednesday December 12

* **Sitemap, Psuedocode and/or Wireframes Ready + Posted**
* Design Strategy Ready: options for use of color, iconography and fonts
* In class code studio

Week 15:

Wednesday December 19

* **Final Project Presentations + Critique**

Friday December 21 by 5pm

* **Final Project Site Due == Links Posted to the Wiki**

Requirements

You will be graded on both the presentation + the project - according to a rubric on concept, execution, visual design, technical sophistication and effective communication - including commented code w/ resources cited!

If your site is still buggy on the day of your presentation - that's okay. You will have until the end of the week to post the link in order to hand in the final iteration.

Final Presentation

Presentation structure

Introduction & context:

What is it that you made?

Why this project?

Tour of a website, app or project:

- overview of different pages and elements utilized
- visual choices - why did you choose this design strategy?
- how you want user to interact with your site?
- if something doesn't work, what is it supposed to do?

Code overview:

- explain your code - structure, html elements, css, javascript
- find a specific part of code that you think is interesting , took you some time, or you're proud of
- what's the logic of the code for this portion between the HTML, CSS and JAvaScript?

If there's transitions, how does it work?

Closing:

- what did you learn?
- what did you wish you did differently?
- how you would take this project further