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("Where's our snow day?")
    } 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

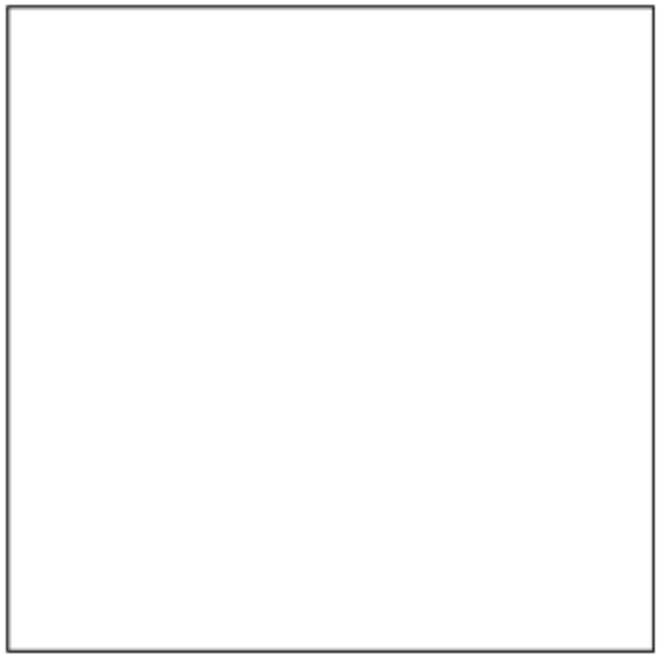
```
function alertMe( ) {
        alert(“It’s me!”);
}
setTimeout(alertMe, 2000);

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

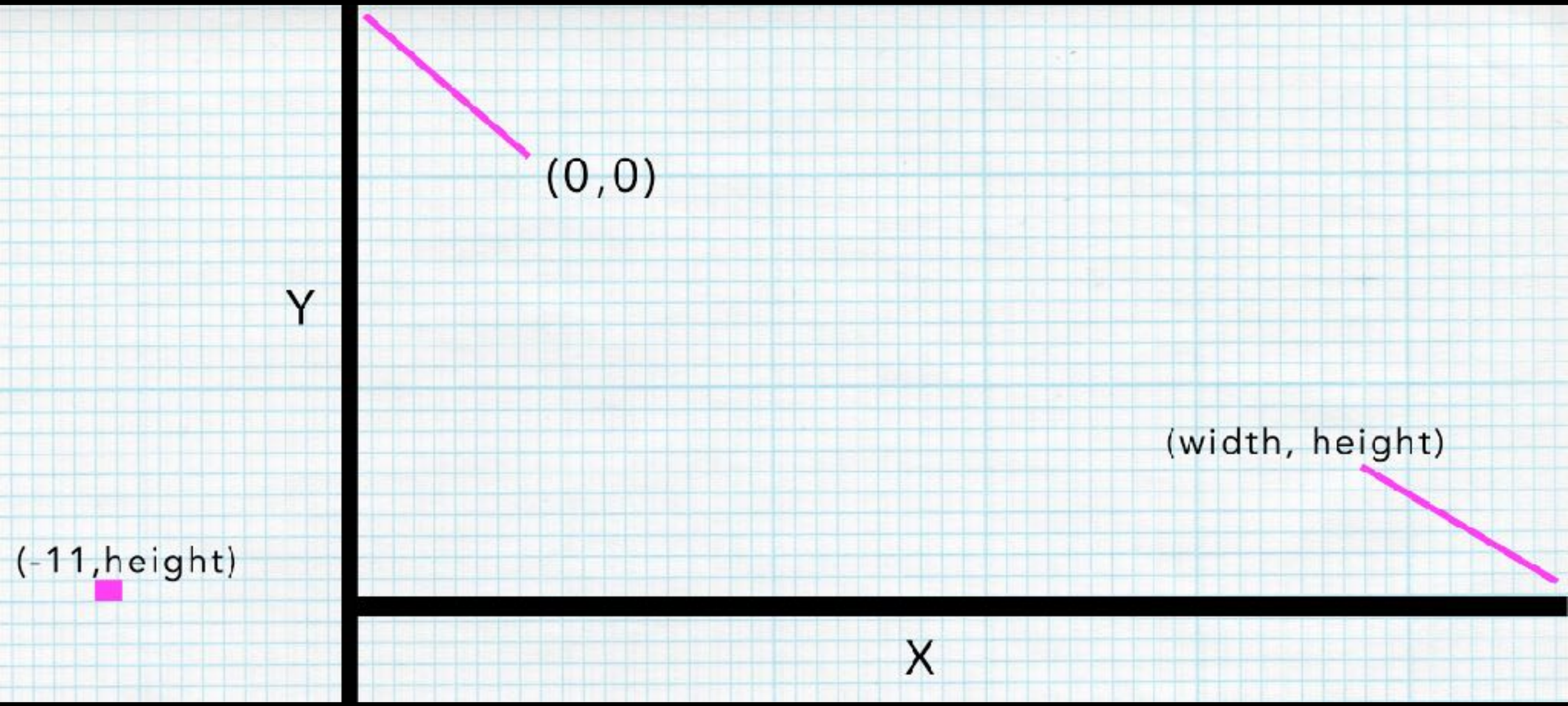
```
function alertMe( ) {
        alert(“It’s me!”);
}
var alertIs = setInterval ( alertMe, 2000);

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

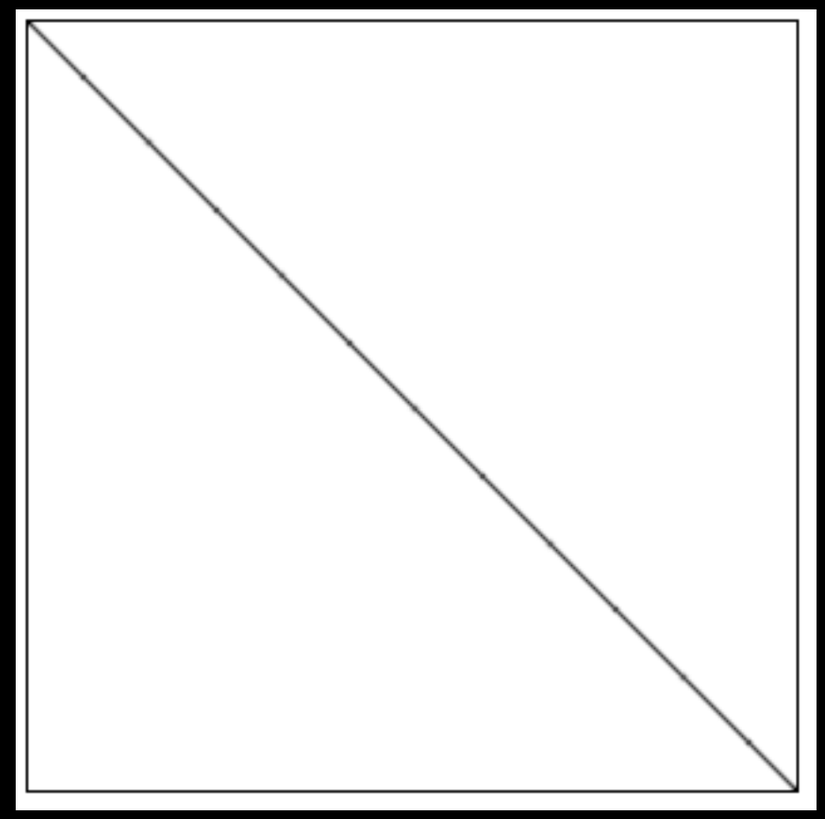


Canvas

+ 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…