

ECS 189
WEB PROGRAMMING

4/19

Weather App, 2017

Davis, CA

April 17 Rain 65	April 18 Rain 65	April 19 Sunny 68	April 20 Sunny 70	April 21 Cloudy 68	➔
------------------------	------------------------	-------------------------	-------------------------	--------------------------	---

Zip code or place

Moving an element

- CSS “position” properties allow you to change where flexbox decided to put items.
- Use as last resort for static designs
- Very handy for allowing Javascript to move stuff!
- I was going to use `position: absolute;`
- Which lets you specify the position of an element within its parent’s box. But there is an easier way!

CSS position property

`position: relative;`

- First, lets flexbox determine position of element; then, we specify offsets (left, right, top, bottom) from that position.
- And, we can specify the offsets using Javascript!

Now we need to change it

- For a change, “left” in CSS corresponds to “left” in Javascript!

```
left = left+10;
steppy.style.left = left+"px";
```

- Marches off the right side – woops!
- To make it disappear as it hits the edge of the box, set the parent container (#range) to have: `overflow: hidden;`

Stop before the end

- Get the width of the parent box so we know when we’re getting ready to hit the end.

```
var container = steppy.parentElement;
var width = container.clientWidth;
```

- Test to see if near far right before moving.

```
if (left < width-225) {
    left = left+10;
}
```

Special characters

- Use unicode encoding for characters that don't appear on the keyboard, eg:

```
<p> 63 &#xb0; </p>
```

...to get 63 degrees. Some people used

🔍 for the search magnifying glass in the last assignment, but it is not supported in all fonts.

Organize code using objects

- We want to organize collections of data and functions that act on that data.
- Organizing data is one way of forcing ourselves to keep our code organized, which is part of the eternal battle against bugs.
- Since objects can contain methods (functions), we can also use objects to organize the functions as well.
- In **object-oriented programming**, almost all the code is inside objects.

A forecast object

```
var forecast = {  
  "id": 1,  
  "description": "sunny",  
  "temp": 66  
};
```

- We're defining the object by giving a literal – a text representation of its contents – and putting those contents into a variable.

Literal

- A **literal** in a computer language is the string used for writing down a fixed value.
 - "2" is a number literal
 - "true" is a Boolean literal
 - "cow" is a string literal
 - {"cow":2} is an object literal

A question object

```
var forecast = {  
  "id": 1,  
  "description": "sunny",  
  "temp": 66  
};
```

- We access the properties as usual, with the dot, eg.
forecast.id == 1; /* this will be true */

A question object

```
var forecast = {  
  "id": 1,  
  "description": "sunny",  
  "temp": 66  
};
```

- We can also access the properties with brackets
forecast[id] == 1; /* this will be true */

What are objects “really”?

- A Javascript object is ...
- a Python dictionary!
- In C, you'd use a hashtable (or some other Dictionary data structure that lets you look up data using a string).
- How is this different from a struct?

What are objects “really”?

- A Javascript object is ...
- a Python dictionary!
- In C, you'd use a hashtable (or some other Dictionary data structure that lets you look up data using a string).
- How is this different from a struct?
 - A struct has a fixed set of properties, stored in an array.
 - You can put in and take out properties of a Javascript object on the fly

For example...

```
var forecast = {  
  "id": 1,  
  "description": "sunny",  
  "temp": 66  
};
```

```
forecast.low = 58;
```

- We give the forecast a property “low”, and put the number 58 into it.
- We can now access “low” just like any other property.

Better way to organize it

```
var forecast = {  
  "id": 1,  
  "description": "sunny",  
  "temp": {  
    "high": 66,  
    "low": 58  
  }  
};
```

- An object one of whose properties is an object

Using a hierarchy of objects

```
forecast.temp.low == 58;  
/* this would be true */
```

- Javascript arrays can also be defined by giving a literal.

```
var arr = [1,2,3]; /* array containing 1,2 and 3 */
```

Objects can contain arrays

```
var forecast = {  
  "id": 1,  
  "description": "sunny",  
  "temp": { "high": 66, "low": 58 },  
  "hourly": {58, 59, 60, 62, 62, 66, 65}  
  // temps noon through 6pm  
};
```

- How to access temperature at 2pm?

Objects can contain arrays

```
var forecast = {  
  "id": 1,  
  "description": "sunny",  
  "temp": { "high": 66, "low": 58},  
  "hourly": {58, 59, 60, 62, 62, 66, 65}  
  // temps noon through 6pm  
};  
  
forecast.hourly[2] == 60; //true!
```

JSON = Object literals

- Data is transmitted between the different computers making up a Web application in a format called JSON.
- The JSON format is a Javascript object literal
- You can use a JSON string to initialize an object

JSON.parse method to make objects

```
cattleJSON = ' {"cow": "herford", \  
  "num": 2 }';  
cattleObj = JSON.parse(cattleJSON);
```

- JSON.parse() takes JSON as input. Produces the corresponding object. What does "parse" mean?

JSON.parse method to make objects

```
cattleJSON = ' {"cow": "herford", \  
  "num": 2 }';  
cattleObj = JSON.parse(cattleJSON);
```

- Note the "\" – lets a string extend over multiple lines.
- The "\n" tells the Javascript interpreter to ignore the newline.

JSON.stringify for obj->JSON

```
cattleObj = {"cow": "herford",  
  "num": 2 };  
cattleJSON = JSON.stringify(cattleObj);
```

- JSON.stringify() takes object as input. Produces the corresponding JSON string.