Objects

object

An object is a variable that stores a collection of properties bundled in curly braces {}.

properties

- Properties are variables belonging to an object; as such, they have a name and a value.
- Properties are child variables of the parent object variable.

key-value pairs

- Properties are name-value pairs, called **key-value** pairs.
- A property name is called a **key**. Unlike "regular" variable names, keys can have spaces.
- The value can be of any data type: string, number, boolean--even object and function.

method

- An object property whose value is a function is called a **method**.

const for objects

As with arrays, we will usually declare with `const`. This gives us the freedom to modify the properties at will, while keeping us in our swim lane, so to speak, by preventing us from changing the data type.

difference between object and array

Arrays have a data type of object, but array items have no name-value pairs, but are rather numbered by position (index), and are bundled in square brackets.

DOM objects

When a web page (DOM) element is brought into JS, it exists in the script as an **object**. The properties of the object are the attributes of the elements. We have already been working with the DOM by getting elements and setting their properties.

1. Open **04.02-Objects.html** and preview it in the browser. This page is just a springboard for checking our console.log output. Remember to switch the file being used from FINAL .js to START .js.

2. In the JS file, declare a new object variable, and assign it some properties in between its curly braces. The key-values are separated by colons. Each property ends with a comma to separate it from the next property. Optionally, a so-called **trailing comma** may come after the last property.

```
const car = {
    make: 'Ford',
    model: 'Mustang GT',
    year: 1999,
    color: 'red',
    condition: 'excellent',
    miles: 123456,
    onRoad: true,
    forSale: false
};
```

Properties of an object are available only to the object, and are referenced by dot-sytnax: `object.property`.

3. Log the whole object, as well as a few properties. In the console, open the arrow to see the properties. Notice that, in contrast to array items, properties are not numbered:

```
console.log(car);
console.log(car.year, car.make, car.model);
```

Properties are added to an existing object by dot syntax: `object.property = value`. Adding a property amounts to declaring a variable *scoped* to the object.

nested objects: objects as object properties

Object properties can be other objects, including arrays.

4. Add two more properties to the car object, one an array, the other a child object: and then access them using "dot.dot" and square bracket syntax

```
car.mpg = {city: 18, hwy: 25};
car.options = ['sun roof', 'CD player', 'leather seats'];

console.log(car);
```

5. Access the new properties, using "dot.dot" for the child object and square brackets for array items:

```
console.log('MPG City:', car.mpg.city);
console.log('Audio:', car.options[1]);
```

Set (change) a property value with dot-syntax in the same way you would add a property:

6. Update some of the property values of the car object.

```
car.miles = 135790;
car.condition = "very good";
car.mpg.city = 17;
car.mpg.hwy = 24;
car.options[0] = "moon roof";
car.forSale = true;

console.log(car);
```

consolidating related properties into child objects

7. Add three properties having to do with the engine:

```
car.cylinders = 8;
car.liters = 4.6;
car.horsepower = 260;
```

Since `cyl` (cylinders), `hp` (horsepower) and `ltr` (liters) all have to do with the engine, we can bundle these into an `engine` property.

8. Retool the object by "bundling" `cyl`, `hp` and `ltr` into an `engine` property. This also lets us abbreviate propeterties without obscuring the meaning:

```
car.engine = { cyl: 8, ltr: 4.5, hp: 270 };
```

9. Log both versions of the duplicate engine properties:

```
console.log(car.horsepower, car.engine.hp); // 260 260
console.log(car.cylinders, car.engine.cyl); // 8 8
console.log(car.liters, car.engine.L); // 4.6 4.6
```

delete keyword

The `delete` keyword is used to remove object properties, as `detete object.propery`.

10. Delete `horsepower`, `cylinders` and `liters` and then log `car` to make sure they've been deleted:

```
delete car.horsepower;
delete car.cylinders;
delete car.liters;

console.log(car);
```

keys can have spaces

While there is generally no upside to having spaces in keys, it is allowed. The key goes in quotes and is accessed with square brackets: `object[key]`.

11. Add two properties with spaces in the keys. Log the `car` object to confirm that they got added:

```
car["consumer reviews"] = 234;
car["star rating"] = 4.7;

console.log(car);
```

toLocaleString()

The `number.toLocaleString()` method is called on a number and returns the number with commas, which converts it to a string.

12. Convert a number with no commas to a "number-like string" with commas:

```
let price = 21500;
let priceStr = price.toLocaleString();
console.log(priceStr, typeof(priceStr));
```

object methods

- When a property value is a function, that property is known as a **method**.
- A method must `return` a value
- A method is called using dot syntax: `object.method()`
- A method can make use of the object's properties, accessing them by referring to the object itself as `this`.

this keyword

The `this` keyword refers to different objects, depending on the context:

- in the global scope, `this` is the Global Window Object
- inside a function, `this` is the object that "owns" the event that calls the function; in the case of a function called by a button click, `this` is the button.
- inside a method, `this` is the object itself.

13. Define a method called `listForSale`. Have it return a "FOR SALE" listing. Refer to various properties using the `this` keyword:

```
car.listForSale = function() {
    return `<strong>FOR SALE!</strong>
    ${this.year} ${this.make} ${this.model},
    only ${car.miles.toLocaleString('en-us')} miles,
    ${this.condition} condition.
```

```
        Loaded with options:<br>${this.options[0]}, ${this.options[1]},
${this.options[2]} and much more.
        MPG: ${this.mpg.hwy} highway, ${this.mpg.city} city.
        Price negotiable.`;
    }
```

13. Call the method in two ways; directly log it and also save it to a variable, and then log the variable:

```
    console.log(car.listForSale());
    let listing = car.listForSale();
    console.log(listing);
```

DOM output of object properties as "Details Page"

In a search application, that page that displays the search results is the "results page". When you click on an individual result, you go to the "details page" where you see more info about that item. Let's make a "details page" for the car, as if it were a search result.

The html and css for this are already done:

```
    <div>
        <img src="images/1999-Ford-Mustang-GT.jpg" id="car-pic">
        <h2 id="car-title">year make model</h2>
        <h3 id="car-mpg">hwy city</h3>
        <h3 id="car-engine">engine L cyl hp</h2>
        <h3 id="car-options">options</h2>
        <hr>
        <h3 id="car-reviews-ratings">reviews ratings</h2>
        <hr>
        <p id="car-listing">listing</p>
    </div>
```

14. Get the elements for displaying the car data:

```
    const carTitle = document.getElementById('car-title');
    const carMPG = document.getElementById('car-mpg');
    const carEngine = document.getElementById('car-engine');
    const carOptions = document.getElementById('car-options');
    const carListing = document.getElementById('car-listing');
    const carReviewsRatings = document.getElementById('car-reviews-
ratings');
    const carPic = document.getElementById('car-pic');
```

There's no button or menu for displaying the car data; it just displays automatically on page load. But still we can have the document call a function when it loads.

15. Have the body `onload` event call an anonymous function when the document is fully loaded:

```
    document.body.onload = function() {
    }
```

16. Output the object properties to the webpage:

```
    document.body.onload = function() {

        carTitle.textContent = `${car.year} ${car.make} ${car.model}`;

        carMPG.textContent = `MPG: ${car.mpg.hwy} hwy - ${car.mpg.city}
city`;

        carEngine.textContent = `Engine: V${car.engine.cyl}
${car.engine.L} L - ${car.engine.hp} horsepower`;

        carOptions.textContent = `Options: ${car.options[0]},
${car.options[1]}, ${car.options[2]}`;

        carReviewsRatings.textContent = `${car['consumer reviews']}
customer reviews - ${car['stars rating']} stars`;

        carListing.innerHTML = car.listForSale();
    }
```

looking up object property by dynamic key

A dynamic key is a variable serving as an object key. Looking up items by dynamic key requires square bracket -- not dot -- syntax.

Next we'll work with an `animals` object with numerous properties, each an individual animal.

17. Open `animals.js` and have a look:

```
    const animals = {

        'American bison': { class: 'mammal', herbivore: true, continent:
'North America' },

        anaconda: { class: 'reptile', herbivore: false, continent: 'South
America' },

        // -- ETC. --
    }
```

- It is an object called `animals` with 18 properties, each an individual animal.

- Each `key` is an animal name, with the two-word keys in quotes.
- Each `value` is an object with four properties:
  - class (string)
  - herbivore (boolean)
  - continent (string)
  - legs (number)

We will write a program where the user chooses an animal from the menu to load its info:

- The user chooses an animal from a menu
- The change event calls a function
- The function gets the choice and saves it to a variable, `chosenAnimal`. The choice is the name of an animal (e.g. 'panda', 'lion')
- The function looks up that animal by dynamic key: `animals[chosenAnimal]`
- The animal data is outputted to the page

The html and css for this are already done, with zebra displayed by default. Both JS files: `animals.js` and `04.02-Objects-FINAL.js` are already imported into the html file.

```
<select id="animals-menu">
    <option id="choose">Choose an animal..</option>
    <option id="American bison">American bison</option>
    -- ETC. --
</select>

<div id="animal">
    <img src="images/zebra.jpg" id="animal-pic">
    <div>
        <p id="animal-name">zebra</p>
        <p id="animal-class">class: mammal</p>
        <p id="continent">continent: Africa</p>
        <p id="herbivore">herbivore: yes</p>
        <p>
    </div>
</div>
```

18. Get the elements for displaying the animal data:

```
const animalsMenu = document.getElementById('animals-menu');
const animalName = document.getElementById('animal-name');
const animalClass = document.getElementById('animal-class');
const continent = document.getElementById('continent');
const herbivore = document.getElementById('herbivore');
const animalPic = document.getElementById('animal-pic');
```

19. Have the menu call a function on 'change' (menu choice):

```
animalsMenu.addEventListener('change', displayAnimalInfo);
```

20. Write the function, starting with saving the menu choice to a variable:

```
function displayAnimalInfo() {

    // get the menu choice, which is an animal name
    let chosenAnimal = animalsMenu.value; // e.g. giraffe, panda
}
```

21. Look up the animal in the `animals` object, using the variable as the key. For this dynamic property accessor, use square brackets, not dot-syntax:

```
let animalObj = animals[chosenAnimal];
```

22. Output the animal properties to their respective tags:

```
animalName.textContent = chosenAnimal;
animalClass.textContent = 'Class: ' + animalObj.class;
continent.textContent = 'Continent: ' + animalObj.continent;
herbivore.textContent = 'Herbivore: ' + animalObj.herbivore;
```

23. Set the souce of the animal image. That completes the function:

```
    animalPic.src = `images/${chosenAnimal}.jpg`;

} // end function
```

END: Lesson 04.02 NEXT: 04.02 LAB