Lesson 04.02 - PROG

Objects

There are four categories of JS objects that interest us, three of which we are already quite familiar with:

- A. Built-in Objects: JS objects that do not pertain to the DOM.
  - Math object, with its methods: floor(), sqrt(), etc. The Math object does not need to be instantiated--just use it
  - Date object, with its methods: getHours(), getDate(), etc. The Date object must be instantiated: let dt = new Date()
- B. DOM objects: JS "versions" of HTML elements: body, div, image, button, select, input, h1, h2, p, etc.
  - DOM objects have properties that "belong" to the object.
  - HTML elements are brought into JS by various methods, viz: getElementById(), querySelector(), querySelectorAll()
  - Properties are accessed via "dot-syntax": pic.src
  - Properties exist as key-value pairs: pic.src = "cat.jpg" pic is the object, src is the key, "cat.jpg" is the value
  - Properties may be specific to particular DOM objects: image.src, checkbox.checked, input.value, select.selectedIndex
  - Other properties are applicable to practically any DOM object: image.id, checkbox.id, select.id, div.id, button.id, input.id
- C. Arrays are objects that store values in numeric order (index): fruits = ['apple', 'banana', 'cherry']
  - The first item is at index 0: fruits[0] // apple
  - Array items are not properties; they are "index-value" pairs
  - Arrays have many methods: push(), pop(), sort(), etc.

This fourth kind of object is what this lesson is all about:

- D. "User-Defined Objects" are made up by the programmer:
  - User-defined objects consist of properties in curly braces: const pet = {type: "Cat", name: "Fluffy", age: 3, cute: true}
  - As with DOM objects, properties are accessible via dot-syntax: console.log(pet.name); // Fluffy
  - In Web applications, datasets often exist as arrays of objects.

1. Open 04.02-Objects.html, and preview it in the browser.
2. Click the Get Car Info button to display car object property values.
3. Choose an animal from the menu; we get animal object property values.
4. In 04.02-Objects.html, switch from importing FINAL.js to PROG.js.
5. Switch to 04.02-Objects.js.

6. Declare a const called sportscar, and set it equal to a pair of curly braces.

```
const sportscar = {};
```

We use const to protect the data type from being mutated to, say, a string.

7. Try to change sportscar to a string. We get an error:

```
sportscar = "Porsche"; // ERROR: assignment to constant variable.
```

An object declared with const cannot be mutated, but its properties can be added, removed and changed as we see fit.

8. Declare another object, this time assigning it properties as key-value pairs. The key and its value are separated by a colon. Properties are separated one from the other by a comma. Values can be any data type: strings, numbers and booleans.

```
const car = {
    category: "sports car",
    make: 'Ford',
    model: 'Mustang GT',
    year: 1998,
    color: 'red',
    condition: 'excellent',
    miles: 123456,
    onRoad: true,
    forSale: false
};
```

9. Log the object and open it up in the Console to check the properties, which have been alphabetized:

```
console.log(car);
```

dot-syntax

Properties are available only to their object.vTo reference a property, use the dot-sytnax: **car.year** returns the value **1998**.

10. Get some car properties, which returns the values:

```
console.log(car.category); // sports car
console.log(`For Sale: ${car.year} ${car.make} ${car.model}`); // For
Sale: 1999 Ford Mustang GT
```

```
console.log(car.year); // 1998
console.log(car.miles); // 123456
console.log(car.onRoad); // true
```

setting property values

To set a value, get the property and set it to a new value: **car.year = 1999** changes the year to **1999**. Inside the curly braces, key-value pairs are separated by a colon, but when a value is changed later, an equal sign is used instead:

11. Set some car properties; this changes the value:

```
car.category = "muscle car";
car.condition = "very good";
car.miles = 123567;
car.year = 1999;
forSale = true;
```

12. Log the whole object to see the changed values:

```
console.log(car);
```

adding properties

Properties are added to an existing object the same way a property value is changed: object.key = value

13. Add three more properties to the car object: a string, a number and a boolean:

```
car.transmission = "manual";
car.doors = 2;
car.convertible = true;
console.log(car);
```

arrays and objects as properties

Properties can be of any data type, including arrays and even other objects.

14. Add two more properties: an array and an object:

```
car.options = ['sun roof', 'CD player', 'leather seats'];
car.mpg = {city: 18, hwy: 24};
console.log(car);
console.log(car.options); // ['sun roof', 'CD player', 'leather
seats']
console.log(car.mpg); // {city: 18, hwy: 24}
```

15. Access array items using square bracket syntax:

```
console.log(car.options); // ['sun roof', 'CD player', 'leather
seats']
console.log(car.options[2]); // leather seats
```

16. Access child objects using "dot-dot" syntax:

```
console.log(car.mpg.city); // 18
console.log(car.mpg.hwy); // 24
```

bundling related properties into one

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

```
car.cylinders = 8;
car.liters = 4.6;
car.horsepower = 260;
console.log(car);
```

It works, but since these properties all pertain to the engine, better might be to make an engine property and assign it an object having the three properties:

18. Bundle cylinders, liters and horsepower into an engine property. Since "engine" will be included in any reference to these properties, we can abbreviate the names without causing any confusion:

```
car.engine = {cyl: 8, ltr: 4.6, hp: 260};
```

object property with array as its value:

~~~
```js
car.options = ['sun roof', 'CD player', 'leather seats'];
console.log(car);
```
~~~

19. Access the engine properties using "dot-dot" syntax:

```
car(car.engine.cyl); // 8
car(car.engine.ltr); // 4.6
```

```
car(car.engine.hp); // 260
```

deleting properties

The delete keyword, preceding a property reference, removes that property. Now that car has an engine property, we can delete horsepower, cylinders and liters.

20. Delete the horsepower, cylinders and liters properties:

```
delete car.horsepower;
delete car.cylinders;
delete car.liters;
console.log(car);
```

square bracket syntax for keys

keys with spaces

Keys are essentially variable names, but unlike "regular" variables, keys can have spaces. This may be a good option for a two-word key that we prefer to keep readable as such, as opposed to using camelCase. The space means we cannot use dot-syntax. Instead, the keys go in quotes, inside square brackets:

21. Add a few properties of two words each:

```
car["top speed"] = 149;
car["consumer reviews"] = 234;
car["stars rating"] = 4.7;
console.log(car);
console.log(car["top speed"]);
```

22. Try to access "top speed" via dot syntax. With or without quotes, we get an error:

```
// car.top speed; // ERROR
// car."top speed"; // ERROR
```

dynamic variable keys

A key itself can be dynamic, that is a variable. For dynanmic keys, dot syntax won't work. Use square brackets, instead.

23. Declare a carKey variable, and set it equal to one of the keys, as a string:

```
let carKey = "model";
```

24. Using dot-syntax, try to get the car model as carKey. We get an error, because car has no carKey property:

    ```
    console.log(car.carKey); // ERROR
    ```

25. To make car.carKey be understood as car.model, use square brackets:

    ```
    console.log(car[carKey]); // Mustang GT
    ```

26. To set a multi-word key value, also use square brackets:

    ```
    car["consumer reviews"] = 248;
    console.log(car["consumer reviews"]); // 248
    ```

toLocaleString()

The toLocaleString() method is called on a number and returns the number with commas, converting it to a string in the process.

27. Get the six-digit mileage, and output it with a comma:

    ```
    console.log(car.miles.toLocaleString()); // 123,567
    ```

object methods

- A method is a function that is scoped to -- belongs to -- an object
- A method is a property of an object, where the value is a function
- To make a method, set a key equal to an anonymous function
- Inside a method, all other properties are available via the 'this' keyword
- A method must return a value: call a method, gets the return value

28. Assign the car object a method, called listForSale. It returns a FOR SALE listing, concatenated from numerous properties referenced by 'this' syntax:

    ```
    car.listForSale = function() {
    return `FOR SALE! ${this.year} ${this.make} ${this.model},
    only ${this.miles.toLocaleString('en-us')} miles, ${this.condition}
    condition.
    Loaded: ${this.options[0]}, ${this.options[1]}, ${this.options[2]},
    much more.
    MPG: ${this.mpg.hwy} highway, ${this.mpg.city} city. Best Offer!`;
    }
    ```

29. Log a call to the method. We get the classified listing:

```
console.log(car.listForSale());
```

30. Declare a variable, set equal to a call to the method, and then log it:

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

outputting property values to the DOM

31. Get the elements where the car data is displayed:

```
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 carReviewsRating = document.getElementById('car-reviews-
ratings');
const carPic = document.getElementById('car-pic');
```

32. Get the button, and have it call the showCarInfo function when clicked:

```
const btn = document.querySelector('button');
btn.addEventListener("click", showCarInfo);
```

33. Define the showCarInfo function:

```
function showCarInfo() {

// 34. Hide the button, since it only gets clicked once:
this.style.display= "none";

// 35. Output the title as "year make model":
carTitle.textContent = `${car.year} ${car.make} ${car.model}`;

// 36. Output the properties of the mpg property:
carMPG.textContent = `MPG: ${car.mpg.hwy} hwy - ${car.mpg.city} city`;

// 37. Output the properties of the engine property:
carEngine.textContent = `Engine: V${car.engine.cyl} ${car.engine.ltr}
L ${car.engine.hp} hp - ${car.model}`;
// 38. Output the options, one array item at a time:
carOptions.textContent = `Options: ${car.options[0]},
${car.options[1]}, ${car.options[2]}`;

// 39. Output the two-word properties, using square bracket syntax:
carReviewsRating.textContent = `${car['consumer reviews']} consumer
reviews, ${car['star rating']} consumer reviews`;

// Output the FOR SALE listing, which is returned by the listForSale()
method:
carListing.textContent = `${car.listForSale()}`;

} // end function showCarInfo()
```

arrays of objects

In web applications, data is often stored as arrays of objects or objects with child objects for properties. For this animal menu part of the lesson, we will work with an extenal file, animals.js, which contains an array of animal objects.

40. Open animals.js. It contains two big variables, an object called animalsObj, and an array by the name of animalsArr.

- animalsObj consists of 20 properties:
    - Each key is an animal name, with 2-word names in quotes.
    - Each value is an object with 3 properties:
        - class (value is string)
        - herbivore (value is boolean)
        - continent (value is string)
- animalsArr has 20 items, each an object
    - The objects are not key-values
    - The animal name is a property--not a key
    - Each array item object has 4 properties:
        - name (value is string)
        - class (value is string)
        - herbivore (value is boolean)
        - continent (value is string)

We will use animalsObj for this exercise:

41. Get the DOM elements for the animals:

```
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');
```

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

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

43. Define the function that runs when a menu choice is made:

```
function showAnimalInfo() {

    // 44. The menu value is an animal, such as "giraffe" or "panda".
Save it to a variable, animal:
    let animal = animalsMenu.value; // e.g. giraffe, panda

    // 45. Look up the animal in the animals object, using the string
variable as the dynamic key. For this, we use square brackets--not
dot-syntax:
    let animalObj = animals[animal];

    // 46. Output the property values to their respective tags:
    animalName.textContent = animal;
    animalClass.textContent = 'Class:' + animalObj.class;
    continent.textContent = 'Continent:' + animalObj.continent;
    herbivore.textContent = 'Herbivore:' + animalObj.herbivore;

    // 47. Set the souce of the animal image:
    animalPic.src = `images/${animal}.jpg`;
}
```

CHALLENGE:

Make another function that produces the same DOM output. BUT rather than get data from the animals object, uses animalsArr, which is the array of objects.

~~~
```js
animalsMenu.addEventListener('change', showAnimalInfo2);

function showAnimalInfo2() {

    // 48. The menu animals are in the same order as the array animals, so
get the index of the chosen animal:
    let indx = animalsMenu.selectedIndex;

    // 49. Look up the animal in the array. Subtract one from the index,
since the second menu choice is the first animal:
~~~

```
    let animalObj = animalsArr[indx-1];
    console.log(animalObj); // {name: 'giraffe', class: 'mammal',
herbivore: true, continent: 'Africa'}

    // 50. Output the name of the animal:
    animalName.textContent = animalObj.name; // giraffe

    // 51. Output the class and continent:
    animalClass.textContent = 'Class: ' + animalObj.class; // mammal
    continent.textContent = 'Continent: ' + animalObj.continent;

    // 52. Output the animal image:
    animalPic.src = `images/${animalObj.name}.jpg`;

    // converting boolean to user-friendly output

    // For the Herbivore part, we don't want to literally output 'true' or
'false'. Better would be a more user-friendly "Yes" or "No". For this we
need conditional logic that sets a string to "Yes" or "No", depending on
the value of the boolean:

    // 53. Do a ternary that sets a variable to "Yes" or "No":
    let yesNo = animal.herbivore ? "Yes" : "No";

    // 54. Output Herbivore: Yes (for giraffe) or Herbivore: No (for lion:
    herbivore.textContent = 'Herbivore: ' + yesNo;

}
```
```
```