

WEB230: JavaScript 1

Module 3: Data Structures

Terminology

We use three kinds of brackets in JavaScript. The names of these are often confusing.

- `()` - Parentheses
 - in arithmetic these are called brackets
- `[]` - Brackets (Square Brackets)
- `{ }` - Braces (Curly Braces)

Data Sets (Arrays)

- Array, set of data
- Written between square brackets
- Values are comma separated
- Spaces inside square brackets are optional
- To access a value use the variable name followed by square brackets enclosing the index
- The first index is 0 (not 1)

```
let listOfNumbers = [2, 3, 5, 7, 11];  
  
console.log(listOfNumbers[2]); // 5  
  
console.log(listOfNumbers[2 - 1]); // 3
```

Properties

- Access "Property" of a value
- Almost all values have properties
 - except `null` and `undefined`
- These properties can contain values or functions
- Access properties with dot or square brackets
 - `value.x` - x is the name of the property
 - `value[x]` - x can be an expression

Properties Continued ...

```
let myName = 'Billy Eilish';  
let propName = 'length';  
console.log(myName.length);  
console.log(myName['length']);  
console.log(myName[propName]);
```

Methods

- String and Array objects contain a number of properties that contain functions

```
let doh = "Doh";  
  
console.log(typeof doh.toUpperCase);  
// → function  
  
console.log(doh.toUpperCase());  
// → DOH
```

Methods Continued ...

- Every string has the `toUpperCase` and `toLowerCase` properties
- Properties that contain a function value are called **Methods**
- the `push` method allows us to add values to the end of an array
- the `pop` method does the opposite and removes the value at the end
- an array of strings can be flattened to a single string with the `join` method


```
let mack = [];  
  
mack.push("Mack");  
mack.push("the", "Knife");  
console.log(mack);  
// → ["Mack", "the", "Knife"]  
  
console.log(mack.join(" "));  
// Mack the Knife  
  
console.log(mack.pop());  
// → Knife  
  
console.log(mack);  
// → ["Mack", "the"]
```

Objects

- Values of the type `object` are arbitrary collections of properties
- We can add or remove properties as we please
- One way to create them is using brace notation (curly braces)

```
// Object representing my car
let myCar = {
  // properties describing my car
  make: "Ford",
  model: "Mustang",
  year: 1969
};

console.log(myCar.make);
// → "Ford"

console.log(myCar.year);
// → 1969

console.log(myCar.color);
// → undefined
```

Objects Continued ...

- the `delete` operator is a unary operator that will remove a property from an object

```
delete myCar.year;  
console.log(myCar.year);  
// → undefined
```

Objects Continued ...

- the `in` operator is a binary operator that will tell you if a property exists in an object

```
console.log("color" in car); // → false
```

Mutability

- With objects, the content of a value can be modified by changing its properties

```
let object1 = {value: 10};  
let object2 = object1;  
let object3 = {value: 10};  
  
console.log(object1 === object2);// true  
console.log(object1 === object3);// false  
  
object1.value = 15;  
console.log(object2.value);// 15  
console.log(object3.value);// 10
```

Mutability Continued ...

- The object1 and object2 variables grasp the same object
 - changing object1 also changes the value of object2
- The variable object3 points to a different object
 - which initially contains the same properties as object1 but lives a separate life

Mutability Continued ...

- When comparing objects, JavaScript's `===` (or `==`) operator
 - Will return true only if both objects are **the same** object
 - Comparing different objects will return false, even if they have identical contents

Array Loops

We can loop through the elements of an array like this:

```
const myPets = ['dog', 'cat', 'rat', 'snake'];  
for(let i=0; i<myPets.length; i++) {  
    console.log(myPets[i]);  
}
```

`for...in` Loop

This is such a common task that some special loops were created, like the `for...in` loop:

```
const myPets = ['dog', 'cat', 'rat', 'snake'];  
for(let i in myPets) {  
  console.log(myPets[i]);  
}
```

Note: the `for...in` loop does not guarantee that the array will be processed in order.

`for...in` Loop on Objects

The `for...in` loop was designed for objects.

```
let car = {  
  make: "Ford",  
  model: "Mustang",  
  year: 1967  
}  
  
for(let prop in car){  
  console.log(prop, car[prop]);  
}
```

`for...of` Loop

The `for...of` loop was designed specifically for arrays:

```
const myPets = ['dog', 'cat', 'rat', 'snake'];  
for(let pet of myPets) {  
  console.log(pet);  
}
```

Note: The `for...of` loop will access array elements in order.

Further Arrayology

- We also have methods `shift` and `unshift` to add and remove from the beginning of an array

```
let todoList = ["homework"];  
let task = "walk dog";  
  
// push a task onto the end of the todo list  
todoList.push(task);  
  
// get and remove the first element  
todoList.shift();  
  
// add a task to the front of the list  
todoList.unshift(task);
```

Arrays Continued ...

- `indexOf` finds the position of a value in an array
- `lastIndexOf` begins searching at the **end**
- Both of these can accept an optional second argument that indicates where to start searching

```
let list = [1,2,3,4,3,2,1];
```

```
list.indexOf(3); // 2
```

```
list.lastIndexOf(3); // 4
```

```
list.indexOf(3,3); // 4
```

Arrays Continued ...

- `.slice()` will create a new array with a segment of the array copied
- takes a start and end index
- When the end index is not given, it will go to the end of the array
- Negative arguments will count from the end of the array

```
let list = [1,2,3,4,3,2,1];  
list.slice(2, 4); // → [3, 4]  
  
list.slice(4); // → [3, 2, 1]
```

Arrays Continued ...

- `concat` can be used to glue arrays together
- Similar to what the `+` operator does for strings

```
let letters = ["a", "b", "c", "d", "e"];  
let numbers = [1,2,3,4,3,2,1];  
  
letters.concat(numbers);  
// → ["a", "b", "c", "d", "e", 1, 2, 3, 4, 3, 2, 1]
```


Strings and Their Properties

- Values of type string, number, and boolean are immutable
 - can't be changed in place
- strings have a number of methods
 - For example `slice`, `indexOf`, and `trim`

```
let nut = "coconuts";  
  
nut.slice(4, 7); // → "nut"  
nut.indexOf("u"); // → 5
```

Strings Continued ...

- `trim()` removes whitespace from the start and end of a string
 - whitespace = space, newline, tab, and similar

```
let nut = " \t coconuts \t \n ";  
nut.trim();  
// → "coconuts"
```

Strings Continued ...

- `charAt()` return an individual character
- Or use square brackets like you'd do for an array

```
let string = "abc";  
  
string.length; // → 3  
  
string.charAt(0); // → a  
  
string[1]; // → b
```

Rest Parameters

- Sometimes it is useful for functions to take any number of arguments.

```
function max(...numbers) {  
    let result = -Infinity;  
    for (let number of numbers) {  
        if (number > result) {  
            result = number;  
        }  
    }  
    return result;  
}  
console.log(max(4, 1, 9, -2));  
// → 9
```

Rest Parameters Continued ...

- This is called a "Rest Parameter"
- All the values are assigned to an Array with the given name

Spread Operator

- Similarly, we can spread the values of an array into individual values, in an array:

```
let words = ["never", "fully"];  
console.log(["will", ...words, "understand"]);  
// → ["will", "never", "fully", "understand"]
```

Or in a function call:

```
let nums = [4, 56, 7, 6, 54, 43, 4, 6, 7, 7];  
let max = Math.max(...nums);
```

The Math Object

- The Math object is a container to group a bunch of related functionality
- There is only one Math object
- It provides a namespace so that all these functions and values do not have to be global variables

```
Math.max(2, 4); // → 4
```

```
Math.min(2, 4); // → 2
```

```
Math.sqrt(4);    // → 2
```

```
Math.PI;    // → 3.141592653589793
```

```
Math.E;    // → 2.718281828459045
```

```
// produce a random number between 0 and 1
```

```
Math.random();    // → 0.36993729369714856
```

```
// produce a whole random number between 1 and 10 inclusive
```

```
Math.floor(Math.random() * 10 + 1);    // → 4
```


Destructuring

```
let person = {name: "Faraji", age: 23, gender: 'M'};  
let {name} = person;  
console.log(name);  
// → Faraji
```

- Works with arrays too:

```
let myPets = ['dog', 'cat', 'gerble', 'pig'];  
let [firstPet, secondPet] = myPets;  
console.log(firstPet, secondPet);
```

JSON

- Often we want to store data to a file or send it to another computer
- We can't send JavaScript arrays or objects as is
- **JSON** is a text notation for JavaScript values

```
{  
  "make": "Ford",  
  "model": "Edge",  
  "year": 2012  
}
```

JSON Continued ...

- JavaScript has functions `JSON.stringify` and `JSON.parse` to convert to and from JSON
 - `JSON.parse` - converts a JSON string to a JavaScript object
 - `JSON.stringify` - converts a JavaScript object to a JSON string

Summary

- Objects and arrays (which are a specific kind of object) provide ways to group several values into a single value
- Most values in JavaScript have properties, the exceptions being `null` and `undefined`
- Properties are accessed using dot notation or square bracket notation

```
value.propName  
value[ "propName" ]
```

Summary Continued ...

- **Objects** tend to use names for their properties and store a fixed set of them
- **Arrays** usually contain varying numbers of conceptually identical values and use numbers as the names of their properties (starting from 0)
- Methods are functions that live in properties and usually act on the value they are a property of

