Agenda

- Recap of Data Structures/Types
- Introduction to JavaScript Objects
- JavaScript Objects
- JavaScript Methods
- Sets
- Maps
- Json

Recap of JS Data Types

In the preceding lectures we discussed about JS strings, numbers and booleans.

JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

```
let carName1 = "Volvo XC60";   // Using double quotes
let carName2 = 'Volvo XC60';   // Using single quotes
```

JavaScript Numbers

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

```
let x1 = 34.00;     // Written with decimals
let x2 = 34;        // Written without decimals
```

JavaScript Booleans

Booleans can only have two values: `true` or `false` .

```
let x = 5;
let y = 5;
let z = 6;
(x == y)       // Returns true
(x == z)       // Returns false
```

JavaScript Arrays

JavaScript arrays are written with square brackets. Array items are separated by commas.

The following code declares (creates) an array called `cars` , containing three items (car names):

```
const cars = ["Saab", "Volvo", "BMW"];
```

Now about the next data-type:

JavaScript object

JavaScript object is a non-primitive data-type that allows you to store multiple collections of data.

Note: If you are familiar with other programming languages, JavaScript objects are a bit different. You do not need to create classes in order to crea objects.

Here is an example of a JavaScript object.

```
// object
const student = {
    firstName: 'ram',
    class: 10
};
```

Here, `student` is an object that stores values such as strings and numbers.

JavaScript Object Declaration

The syntax to declare an object is:

```
const object_name = {
   key1: value1,
   key2: value2
}
```

Here, an object `object_name` is defined. Each member of an object is a **key: value** pair separated by commas and enclosed in curly braces `{}` .

For example,

```
// object creation
const person = {
    name: 'John',
    age: 20
};
console.log(typeof person); // object
```

You can also define an object in a single line.

```
const person = { name: 'John', age: 20 };
```

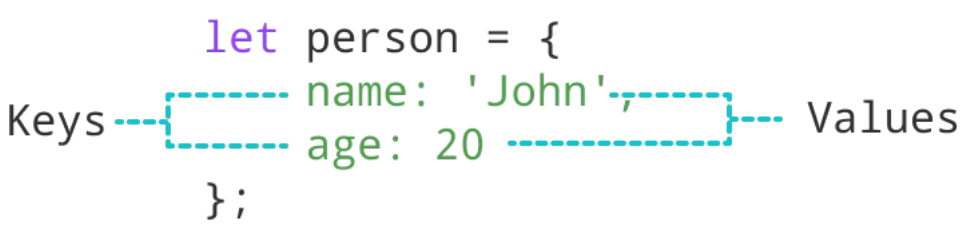
In the above example, `name` and `age` are keys, and `John` and `20` are values respectively.

JavaScript Object Properties

In JavaScript, "key: value" pairs are called **properties**. For example,

```
let person = {
    name: 'John',
    age: 20
};
```

Here, `name: 'John'` and `age: 20` are properties.



JavaScript object properties

Accessing Object Properties

You can access the **value** of a property by using its **key**.

1. Using dot Notation

Here's the syntax of the dot notation.

```
objectName.key
```

For example,

```
const person = {
    name: 'John',
    age: 20,
};

// accessing property
console.log(person.name); // John
```

2. Using bracket Notation

Here is the syntax of the bracket notation.

```
objectName["propertyName"]
```

For example,

```
const person = {
    name: 'John',
    age: 20,
};

// accessing property
console.log(person["name"]); // John
```

JavaScript Nested Objects

An object can also contain another object. For example,

```
// nested object
const student = {
    name: 'John',
    age: 20,
    marks: {
        science: 70,
        math: 75
    }
}

// accessing property of student object
console.log(student.marks); // {science: 70, math: 75}

// accessing property of marks object
console.log(student.marks.science); // 70
```

In the above example, an object `student` contains an object value in the `marks` property.

JavaScript Object Methods

In JavaScript, an object can also contain a function. For example,

```
const person = {
    name: 'Sam',
```

```
    age: 30,
    // using function as a value
    greet: function() { console.log('hello') }
}

person.greet(); // hello
```

Here, a function is used as a value for the greet key. That's why we need to use `person.greet()` instead of `person.greet` to call the functic inside the object.

A JavaScript **method** is a property containing a function declaration. For example,

```
// object containing method
const person = {
    name: 'John',
    greet: function() { console.log('hello'); }
};
```

In the above example, a `person` object has two keys (`name` and `greet`), which have a string value and a function value, respectively.

Hence basically, the JavaScript **method** is an object property that has a function value.

Accessing Object Methods

You can access an object method using a dot notation. The syntax is:

```
objectName.methodKey()
```

You can access property by calling an **objectName** and a **key**. You can access a method by calling an **objectName** and a **key** for that method alor with `()` . For example,

```
// accessing method and property
const person = {
    name: 'John',
    greet: function() { console.log('hello'); }
};

// accessing property
person.name; // John

// accessing method
person.greet(); // hello
```

Here, the `greet` method is accessed as `person.greet()` instead of `person.greet` .

If you try to access the method with only `person.greet` , it will give you a function definition.

```
person.greet; // ƒ () { console.log('hello'); }
```

JavaScript Built-In Methods

In JavaScript, there are many built-in methods. For example,

```
let number = '23.32';
let result = parseInt(number);

console.log(result); // 23
```

Here, the `parseInt()` method of Number object is used to convert numeric string value to an integer value.

Adding a Method to a JavaScript Object

You can also add a method in an object. For example,

```
// creating an object
let student = { };

// adding a property
student.name = 'John';

// adding a method
student.greet = function() {
    console.log('hello');
}

// accessing a method
student.greet(); // hello
```

In the above example, an empty `student` object is created. Then, the `name` property is added. Similarly, the `greet` method is also added. In th way, you can add a method as well as property to an object.

JavaScript this Keyword

To access a property of an object from within a method of the same object, you need to use the `this` keyword. Let's consider an example.

```
const person = {
    name: 'John',
    age: 30,

    // accessing name property by using this.name
    greet: function() { console.log('The name is' + ' ' + this.name); }
};

person.greet();
```

Output

```
The name is John
```

In the above example, a `person` object is created. It contains properties (`name` and `age`) and a method `greet` .

In the method `greet` , while accessing a property of an object, `this` keyword is used.

In order to access the **properties** of an object, `this` keyword is used following by `.` and **key**.

Note: In JavaScript, `this` keyword when used with the object's method refers to the object. `this` is bound to an object.

However, the function inside of an object can access it's variable in a similar way as a normal function would. For example,

```
const person = {
    name: 'John',
    age: 30,
    greet: function() {
        let surname = 'Doe';
        console.log('The name is' + ' ' + this.name + ' ' + surname); }
};

person.greet();
```

Output

```
The name is John Doe
```

JavaScript Sets

A JavaScript Set is a collection of unique values.

Each value can only occur once in a Set.

A Set can hold any value of any data type.

You can create a JavaScript Set by:

- Passing an Array to `new Set()`
- Create a new Set and use `add()` to add values
- Create a new Set and use `add()` to add variables

---

new Set()

Pass an Array to the `new Set()` constructor:

```
// Create a Set
const letters = new Set(["a","b","c"]);
```

Create a Set and add literal values:

```
// Create a Set
const letters = new Set();
// Add Values to the Set
letters.add("a");
letters.add("b");
letters.add("c");
```

Create a Set and add variables:

```
// Create Variables
const a = "a";
const b = "b";
const c = "c";
// Create a Set
const letters = new Set();
// Add Variables to the Set
letters.add(a);
letters.add(b);
letters.add(c);
```

Set Methods

Show All

Method	Description
new Set()	Creates a new Set
add()	Adds a new element to the Set
delete()	Removes an element from a Set
has()	Returns true if a value exists
clear()	Removes all elements from a Set
forEach()	Invokes a callback for each element
values()	Returns an Iterator with all the values in a Set
keys()	Same as values()
entries()	Returns an Iterator with the [value,value] pairs from a Set

Set Property

Show All

Property	Description
size	Returns the number elements in a Set

The `forEach()` method invokes a function for each Set element:

```
// Create a Set
const letters = new Set(["a","b","c"]);

// List all entries
let text = "";
letters.forEach (function(value) {
  text += value;
})
```

The `values()` method returns an Iterator object containing all the values in a Set:

```
// Create an Iterator
const myIterator = letters.values();

// List all Values
let text = "";
for (const entry of myIterator) {
  text += entry;
}
```

A Set has no keys.

`keys()` returns the same as `values()` .

This makes Sets compatible with Maps.

Similarly,

A Set has no keys.

`entries()` returns [value,value] pairs instead of [key,value] pairs.

This makes Sets compatible with Maps:

```
// Create an Iterator
const myIterator = letters.entries();

// List all Entries
let text = "";
for (const entry of myIterator) {
  text += entry;
}
```

Sets are Objects

- `typeof` returns object:
- `instanceof Set` returns true:

```
typeof letters;      // Returns object
```

```
letters instanceof Set;  // Returns true
```

JavaScript Maps

A Map holds key-value pairs where the keys can be any datatype.

A Map remembers the original insertion order of the keys.

A Map has a property that represents the size of the map.

You can create a JavaScript Map by:

- Passing an Array to `new Map()`
- Create a Map and use `Map.set()`

---

new Map()

You can create a Map by passing an Array to the `new Map()` constructor:

```
// Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);
```

You can add elements to a Map with the `set(` method:

```
// Create a Map
const fruits = new Map();

// Set Map Values
fruits.set("apples", 500);
fruits.set("bananas", 300);
fruits.set("oranges", 200);
```

Map Methods

Show All

Method	Description
new Map()	Creates a new Map object
set()	Sets the value for a key in a Map
get()	Gets the value for a key in a Map
clear()	Removes all the elements from a Map
delete()	Removes a Map element specified by a key
has()	Returns true if a key exists in a Map
forEach()	Invokes a callback for each key/value pair in a Map
entries()	Returns an iterator object with the [key, value] pairs in a Map
keys()	Returns an iterator object with the keys in a Map
values()	Returns an iterator object of the values in a Map

Map Property

Show All

Property	Description
size	Returns the number of Map elements

The `set()` method can also be used to change existing Map values:

```
fruits.set("apples", 500);
```

The `get()` method gets the value of a key in a Map:

```
fruits.get("apples");    // Returns 500
```

The `delete()` method removes a Map element:

```
fruits.delete("apples");
```

The `clear()` method removes all the elements from a Map:

```
fruits.clear();
```

The `has()` method returns true if a key exists in a Map:

```
fruits.has("apples");
```

Maps are Objects

- `typeof` returns object
- `instanceof` Map returns true

```
typeof fruits; // Returns object:

fruits instanceof Map; // Returns true:
```

JSON

JSON stands for Javascript Object Notation. JSON is a text-based data format that is used to store and transfer data. For example,

```
// JSON syntax
{
    "name": "John",
    "age": 22,
    "gender": "male",

}
```

In JSON, the data are in **key/value** pairs separated by a comma `,`

JSON was derived from JavaScript. So, the JSON syntax resembles JavaScript object literal syntax. However, the JSON format can be accessed and t
created by other programming languages too.

Note: JavaScript Objects and JSON are not the same. You will learn about their differences later in this tutorial.

JSON Data

JSON data consists of **key/value** pairs similar to JavaScript object properties. The key and values are written in double quotes separated by colon `:` For example,

```
// JSON data
"name": "John"
```

Note: JSON data requires double-quotes for the key.

JSON Object

The JSON object is written inside curly braces `{ }` . JSON objects can contain multiple **key/value** pairs. For example,

```
// JSON object
{ "name": "John", "age": 22 }
```

JSON Array

JSON array is written inside square brackets `[ ]` . For example,

```
// JSON array
[ "apple", "mango", "banana"]

// JSON array containing objects
[
    { "name": "John", "age": 22 },
    { "name": "Peter", "age": 20 }.
    { "name": "Mark", "age": 23 }
]
```

Note: JSON data can contain objects and arrays. However, unlike JavaScript objects, JSON data cannot contain functions as values.

Accessing JSON Data

You can access JSON data using the dot notation. For example,

```
// JSON object
const data = {
    "name": "John",
    "age": 22,
    "hobby": {
    "reading" : true,
    "gaming" : false,
    "sport" : "football"
    },
    "class" : ["JavaScript", "HTML", "CSS"]
}

// accessing JSON object
console.log(data.name); // John
console.log(data.hobby); // { gaming: false, reading: true, sport: "football"}

console.log(data.hobby.sport); // football
console.log(data.class[1]); // HTML
```

We use the `.` a notation to access JSON data. Its syntax is: `variableName.key`

You can also use square bracket syntax `[]` to access JSON data. For example,

```
// JSON object
const data = {
    "name": "John",
    "age": 22
}

// accessing JSON object
console.log(data["name"]); // John
```

JavaScript Objects VS JSON

Though the syntax of JSON is similar to the JavaScript object, JSON is different from JavaScript objects.

JSON	JavaScript Object
The key in the key/value pair should be in double quotes.	The key in the key/value pair can be without double quotes.
JSON cannot contain functions.	JavaScript objects can contain functions.
JSON can be created and used by other programming languages.	JavaScript objects can only be used in JavaScript.

Converting JSON to JavaScript Object

You can convert JSON data to a JavaScript object using the built-in `JSON.parse()` function. For example,

```
// json object
const jsonData = '{ "name": "John", "age": 22 }';

// converting to JavaScript object
const obj = JSON.parse(jsonData);

// accessing the data
console.log(obj.name); // John
```

Converting JavaScript Object to JSON

You can also convert JavaScript objects to JSON format using the JavaScript built-in `JSON.stringify()` function. For example,

```
// JavaScript object
const jsonData = { "name": "John", "age": 22 };

// converting to JSON
const obj = JSON.stringify(jsonData);

// accessing the data
console.log(obj); // "{"name":"John","age":22}"
```

Use of JSON

JSON is the most commonly used format for transmitting data (data interchange) from a server to a client and vice-versa. JSON data are very easy parse and use. It is fast to access and manipulate JSON data as they only contain texts.

JSON is language independent. You can create and use JSON in other programming languages too.

Interview Questions

> How to access keys in an object?

```
let user = {
  name: "Piyush",
  age: 24,
};

for (let key in user) {
  alert( key );  // name, age
  alert( user[key] ); // Piyush, 24
}
```

> What's the output of the following code snippet?

```
const settings = {
  username: 'lydiahallie',
  level: 19,
  health: 90,
};

const data = JSON.stringify(settings, ['level', 'health']);
console.log(data);// "{"level":19, "health":90}"
```

The second argument of `JSON.stringify` is the *replacer*. The replacer can either be a function or an array, and lets you control what and how th values should be stringified.

If the replacer is an *array*, only the property names included in the array will be added to the JSON string. In this case, only the properties with th names `"level"` and `"health"` are included, `"username"` is excluded. ** `data` **is now equal to `"{"level":19, "health":90}"` .

If the replacer is a *function*, this function gets called on every property in the object you're stringifying. The value returned from this function will be th value of the property when it's added to the JSON string. If the value is `undefined` , this property is excluded from the JSON string.

Thank you