Objects

Objects and JSON

Table of Contents

Objects and Properties

Objects in JS

What is an Object?

- An object is a **collection of properties**, and a property is a "**key: value**" pair, where **key** is a string and **value** can anything.
- A field's **value** can be a **function**, in which case it is known as a **method**
- An object is a **reference data type**

Defining an Object

- You define (and create) a JavaScript object with an **object literal**:

```
let person = { firstName:"John", lastName:"Doe", age:50 };
```

- Spaces and line breaks are not important. An object definition can span multiple lines:

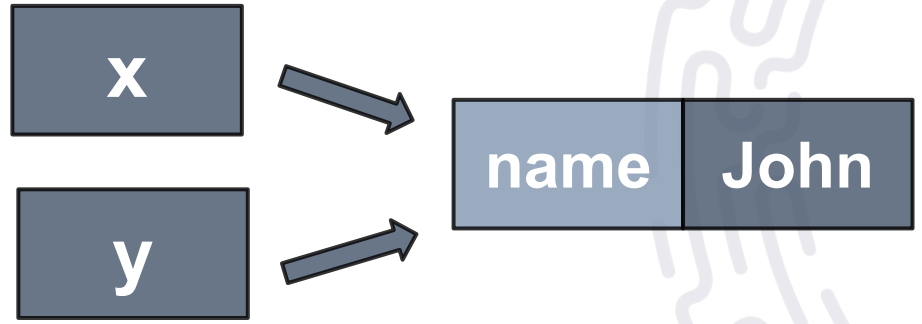
```
let person = {
    firstName: "John",
    lastName: "Doe",
    age: 50
};
```

Variables Holding References

- The **in-memory value** of a reference type is the **reference** itself (a memory address)

```
let x = {name: 'John'};
```

```
let y = x;
```




```
y.name = "John";
console.log(x.name); // John
```

Object Properties

- A **property** of an object can be explained as a **variable** that is **attached** to the object
- Object properties are basically the same as **ordinary** JavaScript **variables**, except for the **attachment** to objects

Property Name	Property Value
`firstName`	`John`
`lastName`	`Doe`
`age`	`50`

Assigning and Accessing Properties

- Simple dot-notation

```
let person = {};
person.name = "Peter";
console.log(person); // { name: 'Peter' }
```

- Bracket-notation

```
person['age'] = 21;
console.log(person); // { name: 'Peter', age: 21 }
```

Assigning and Accessing Properties

- You can access properties the same way

```
let name = person.name;
console.log(name); // Peter

let age = person['age'];
console.log(age); // 21
```

- **Unassigned properties** of an object are **undefined**

```
console.log(person.lastName); // undefined
```

Object Methods

- Objects can also have **methods**
- Methods are **actions** that can be performed on objects
- Methods are stored in **properties** as **function** definitions

```
let person = {
    firstName: "John",
    lastName: "Doe",
    fullName: function () {
        return this.firstName + " " + this.lastName;
    }
};
console.log(person.fullName()); // John Doe
```

This

- The **this** keyword **refers** to the current **object** the code is being written **inside**

```
const person = {
    firstName: 'John',
    lastName: 'Doe',
    returnThis: function(){
        return this;
    }
}
console.log(person.returnThis());
//{ firstName: 'John', lastName: 'Doe', returnThis: [Function: returnThis] }
```

- This will always **ensure** that the **correct values are used** when a member's **context changes**

Objects in JS

- Objects in JavaScript hold **key-value pairs**:

```
let person = {
    name: {
        first: 'John',
        last: 'Doe'
    },
    fullName: function () {
        return this.name.first + " " + this.name.last;
    },
};
//Continues on the next slide
```

Objects in JS

```
person.age = 21;
/*Object {name: {first: 'John',last: 'Doe'},
          fullName: [Function: fullName],
          age: 21}*/
person['gender'] = 'male';
/*Object {name: {first: 'John',last: 'Doe'},
          fullName: [Function: fullName],
          age: 21,
          gender: 'male'}*/
delete person.gender;
/*Object {name: {first: 'John',last: 'Doe'},
          fullName: [Function: fullName],
          age: 21}*/
```

Comparing Objects

- Two variables, **two distinct objects** with the same properties

```
let fruit = {name: 'apple'};
let fruitbear = {name: 'apple'};
fruit == fruitbear; // return false
fruit === fruitbear; // return false
```

- Two variables, a **single object**

```
let fruit = { name: 'apple' };
let fruitbear = fruit;
// Assign fruit object reference to fruitbear
// Here fruit and fruitbear are pointing to same object
fruit == fruitbear; // return true
fruit === fruitbear; // return true
```

Internal Properties

Internal Properties

- Every object field has **four** properties:
  - **Enumerable** - can access to all of them using a **`for...in`** loop
    - Enumerable property are returned using **`Object.keys`** method
  - **Configurable** - can **modify** the **behavior** of the property
    - You **can delete** only **configurable** properties
  - **Writable** - can **modify** their **values** and update a property just assigning a new value to it
  - **Value**

Object's Non-enumerable Properties

- They won't be in for..in iterations
- They won't appear using Object.keys function
- They are not serialized when using JSON.stringify

```
let ob = {a:1, b:2};
ob.c = 3;
Object.defineProperty(ob, 'd', { value: 4, enumerable: false });
ob.d; // => 4
for( let key in ob ) console.log( ob[key] ); //1 2 3
Object.keys( ob );  // => ["a", "b", "c"]
JSON.stringify( ob ); // => "{a:1,b:2,c:3}"
ob.d; // => 4
```

Object's Non-writable Properties

- Once its value is defined, it is **not possible to change** it using assignments

```
let ob = { a: 1 };
Object.defineProperty(ob, 'B', { value: 2, writable: false });
ob.B; // => 2
ob.B = 10;
ob.B; // => 2
```

- If the non-writable property **contains** an **object**, the **reference** to the object is what is **not writable**, but the **object** itself **can be modified**

Object's Non-configurable Properties

- Once you have defined the property as **non-configurable**, there is only **one behavior** you **can change**
  - If the property is **writable**, you can convert it to non-writable
  - Any other try of definition update will **fail** throwing a TypeError

```
let ob = {};
Object.defineProperty(ob, 'a', { configurable: false, writable: true });
Object.defineProperty(ob, 'a', { enumerable: true }); // throws a TypeError
Object.defineProperty(ob, 'a', { value: 12 }); // throws a TypeError
Object.defineProperty(ob, 'a', { writable: false }); // This is allowed!!
Object.defineProperty(ob, 'a', { writable: true }); // throws a TypeError
delete ob.a; // => false
```

Object Freeze and Seal

```
let cat = { name: 'Tom', age: 5 };
Object.freeze(cat);
cat.age = 10;           // Error in strict mode
cat.gender = 'male';    // Error in strict mode
console.log(cat);       // { name: 'Tom', age: 5 }
```

```
cat = { name: 'Tom', age: 5 };
Object.seal(cat);
cat.age = 10;           // OK
delete cat.age;         // Error in strict mode
console.log(cat);       // { name: 'Tom', age: 10 }
```

Looping Through Objects

Object Keys and Values

```
let course = { name: 'JS Advanced', hall: 'Open Source' };
let keys = Object.keys(course);
console.log(keys);  // [ 'name', 'hall' ]
if (course.hasOwnProperty('name')) {
    console.log(course.name);  // JS Advanced
}
```

```
let values = Object.values(course);
console.log(values); // [ 'JS Advanced', 'Open Source' ]
if (values.includes('JS Advanced')) {
    console.log("Found 'JS Advanced' value");
}
```

For… in Loop

- **for** … **in** - iterates a specified variable over all the enumerable properties of an object

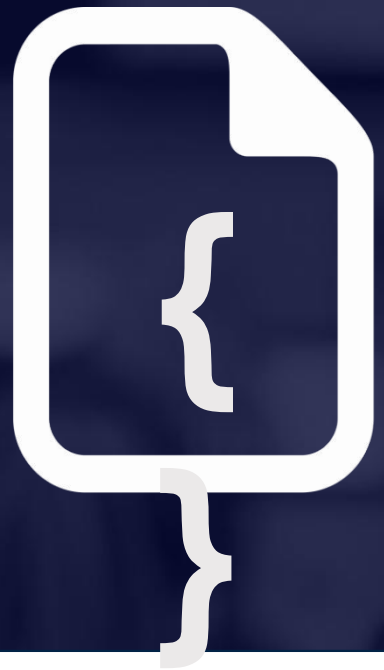
```
let obj = {a: 1, b: 2, c: 3};

for (const key in obj) {

  console.log(`obj.${key} = ${obj[key]}`);

}

// Output:

// "obj.a = 1"

// "obj.b = 2"

// "obj.c = 3"
```

For…of Loop

- The **for...of** statement creates a loop iterating over iterable objects

```
let obj = {a: 1, b: 2, c: 3};
for (const key of Object.keys(obj)) {
  console.log(`obj.${key} = ${obj[key]}`);
}
// "obj.a = 1"
// "obj.b = 2"
// "obj.c = 3"
```

```
for (const val of Object.values(obj)) {console.log(val);}
// 1
// 2
// 3
```



JavaScript Object Notation

JSON

What is a JSON?

- Stands for **J**ava**S**cript **O**bject **N**otation
  - It's a **data** interchange **format**
  - It's **language independent** - syntax is derived from JavaScript object notation syntax, but the JSON format is text only
  - It's **"self-describing"** and easy to understand

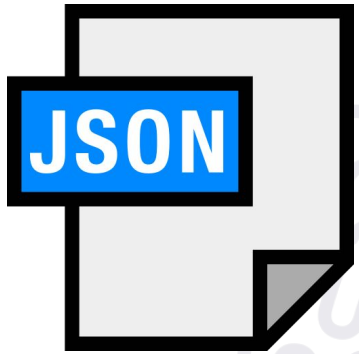
Example: JSON

- This **JSON** syntax defines **`employees`** object - an array of 3 employee records (objects):

```
{
    "employees": [
        { "firstName": "John", "lastName": "Doe" },
        { "firstName": "Anna", "lastName": "Smith" },
        { "firstName": "Peter", "lastName": "Jones" }
    ]
}
```

Syntax Rules

- In JSON:
  - Data is in **name/value** pairs
  - Data is **separated by commas**
  - **Curly braces** hold **objects**
  - **Square brackets** hold **arrays**
  - JSON only takes **double** quotes **""**




```
{
    "employees": [{ "firstName": "John", "lastName": "Doe" }]
}
```

Parsing from Strings

- A common use of JSON is to **read data from a web server**, and **display the data in a web page**
- For simplicity, this can be demonstrated using a string as input

```
let text = '{ "employees" : [' +
    '{ "firstName":"John" , "lastName":"Doe" },' +
    '{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```

- Use the JavaScript built-in function **`JSON.parse()`** to convert the string into a JavaScript object:

```
let obj = JSON.parse(text);
```

Parsing from Strings (2)

- Finally, use the new JavaScript object in your page

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
```

Converting to String

- Use **`JSON.stringify()`** to convert objects into a string:

```
let obj = { name: "John", age: 30, city: "New York" };
let myJSON = JSON.stringify(obj);
console.log(myJSON);
// {"name":"John","age":30,"city":"New York"}
```

- You can do the same for **arrays**

```
let arr = [ "John", "Peter", "Sally", "Jane" ];
let myJSON = JSON.stringify(arr);
console.log(myJSON);
// ["John","Peter","Sally","Jane"]
```

- **`JSON.stringify()`** has the ability to format the string for presentation

Problem: from JSON to HTML Table

- Read a **JSON string**, holding array of JS objects (key / value pairs)
  - Print the objects as **HTML table** like shown below

```
[{"Name":"Tomatoes & Chips","Price":2.35},{"Name":"J&B
 Chocolate","Price":0.96}]
```

```
<table>
    <tr><th>Name</th><th>Price</th></tr>
    <tr><td>Tomatoes &amp;amp; Chips</td><td>2.35</td></tr>
    <tr><td>J&amp;amp;B Chocolate</td><td>0.96</td></tr>
</table>
```

Solution: from JSON to HTML Table

```
function JsonToHtmlTable(json) {
  let arr = JSON.parse(json);
  let outputArr = ["<table>"];
  outputArr.push(makeKeyRow(arr));
  arr.forEach((obj) => outputArr.push(makeValueRow(obj)));
  outputArr.push("</table>");
  function makeKeyRow(arr) { // ToDo }
  function makeValueRow(obj) { // ToDo };
  function escapeHtml(value) { // ToDo };
  console.log(outputArr.join('\n'));
}
```

Summary

- Objects
  - hold **key-value** pairs
  - The key-value pairs in JavaScript objects are called **properties**
  - methods are **actions** that can be performed on objects
- JSON
  - **data** interchange **format**
  - **language independent**
  - **Self-describing**

Questions?

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © Kingsland University – https://kingslanduniversity.com


KINGSLAND
UNIVERSITY
THANK YOU