JS JavaScript Objects

Asst.Prof. Dr. Umaporn Supasitthimethee
ผศ.ดร.อุมาพร สุภสิทธิเมธี

https://github.com/umaporn-sup/1-2566-INT201-basic-array.git

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects

JavaScript: The Definitive Guide, Seventh Edition, by David Flanagan

JavaScript Objects

- JavaScript is an object-based language based on ***prototypes***, rather than being class-based.
- **An object** is an **unordered collection of properties,** name-value pairs where the value may be **data or a function.**
- An object is a **composite value**: it aggregates multiple values (primitive values or other objects) and allows you to store and retrieve those values by name.
- Property names are usually *Strings* or can also be *Symbols.*
- No object may have two properties with the same name.
- JavaScript objects are *dynamic*—properties can usually be added and deleted
- It is possible to create an instance of an “implicit” class without the need to create the class.

JavaScript Object Examples

```
//Simple Object
const student = {
 name: 'Bob',
 age: 32,
 gender: 'male'
}
```

```
//Object Value is array
const profile = {
 id: 123,
 interests: [ 'music', 'skiing']
}
```

```
//Aggregated Object
   const book  = { isbn: 123456789,
                  title: 'JavaScript',
                  author: {
                     firstname: 'Umaporn',
                     lastname: 'Sup'
                  }
                }
```

```
 //Object Value contains function
 const person = {
     id: 1001,
     firstname: 'Somsak',
     lastname: 'Jaidee',
     getFullName: function () {
       return this.firstname + ' ' + this.lastname
     }
 }
 console.log(person.getFullName()) //Somsak Jaidee
```

Ways to create a JavaScript object

1. Object literals

- concise and easy to read
- repeat the code for creating multiple objects of the same type

2. Constructor functions

- create multiple objects of the same type with the same properties and methods
- confusing to use `this` keyword and `new` operator

3. ES6 Classes

- more readable and consistent with other object-oriented languages
- not supported by older browsers

4. Object.create()

- fine-tuned control over the object creation process and inheritance
- complex and verbose

1. Object literals

Simplest form with **object literals**, object literal is a comma-separated list of `{name: value}` pairs.

```
const p1 = {x:10, y: 20}
const p2 = {x:5, y: 10}
```

2. Constructor Functions

Use **new** operator. Objects created using the new keyword and a constructor invocation

```
//constructor
function Point(x, y) {
  this.x = x
  this.y = y
}
const p1 = new Point(1, 2)
const p2 = new Point(2, 4)
console.log(p1) //Point { x: 1, y: 2 }
console.log(p1.x) //1
console.log(p1.y) //2
console.log(p2) //Point { x: 2, y: 4 }
```

The *constructor* method is a special method of a class for creating and initializing an object instance of that class.

When a function is invoked on or through an object, that object is the invocation context or `this` value for the function.

3. ES6 Classes

- ECMAScript 6 (ES6) that provides a syntactic sugar for constructor functions. You use the class keyword to define a class that represents an object type, and then use the new keyword to create instances of the class.

```
class Point {
  constructor(x, y) {
    this.x = x
    this.y = y
  }
  distance(anotherPoint) {
    return Math.sqrt((this.x - anotherPoint.x) ** 2 +(this.y - anotherPoint.y) ** 2)
    //Exponentiation operator (**)
  }
}
```

```
const p1 = new Point(10, 30)//p2={x:10, y:30}
const p2 = new Point(5, 4) //p3={x:5, y:4}
console.log(p1.distance(p2)) // 26.476404589747453
```

4. Object.create()

Use the `Object.create()` function - creates a new object, using an existing object as the prototype of the newly created object.

```
const person = { personId: 101, firstname: 'Somsak', lastname: 'Jaidee' }
const student = Object.create(person)
student.studentId = 651000101
console.log(person)//{ personId: 101, firstname: 'Somsak', lastname: 'Jaidee' }
console.log(student)//{ studentId: 651000101 }
console.log(student.studentId)//651000101
console.log(student.personId)//101
console.log(student.firstname)//Somsak
console.log(student.lastname)//Jaidee
```

Object.create method is very useful when you need to create an object using an existing object as a prototype and can use to create inheritance between objects.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create

Getting, Setting, Creating Object Properties

- To obtain the value of a property, use the dot (.) or square bracket([]) operators

```
const book = {  isbn: 123456789,
              title: "JavaScript",
              author:{
                firstname: "Umaporn",
                lastname: "Sup"
              }
};
```

```
//getting object property
console.log (book.isbn)
console.log(book['title'])
Console.log(book['author']['firstname'])
//setting object property
book.author.firstname = 'Uma'
//create new object property
book['publishedYear']=2000
//book.publishedYear=2000
```

```
object.property
object["property"]
```

```
//output
{
  isbn: 123456789,
  title: 'JavaScript',
  author: { firstname: 'Uma', lastname: 'Sup'
},
  publishedYear: 2000
}
```

- with the[] array notation, the name of the property is expressed as a string.
- Strings are JavaScript data types, so they can be manipulated and created while a program is running.

Defining Methods

- When function is defined as a property of an object, we call that function ***a method***
- Prior to ES6

```
const square ={
    side: 10
    area: function()  { return this.side * this.side},
    //(ES6 Syntax) area (){ return this.side * this.side},
    }
square.area() //100
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects#defining_methods

Using `this` for object references

JavaScript has a special keyword, `this`, that you can use within a method to refer to the current object.

```
const square1 = {
  side: 10
}
const square2 = {
  side: 20
}
function area() {
  return this.side * this.side
}
square1.area = area
square2.area = area
console.log(square1.area()) //100
console.log(square2.area()) //400
```

```
const square1 = {
  side: 10,
  area() {
    return this.side * this.side
  }
}
console.log(square1.area()) //100

const square2 = Object.create(square1)
square2.side = 20
console.log(square2.area()) //400
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects#using_this_for_object_references

Object Passing to functions by reference

- Objects are ***mutable*** and manipulated by reference rather than by value.

```
let point = { x:10, y: 20 }
let newPoint = point
newPoint.x = 30
console.log (point) //{x:30, y:20)
```

Object Passing to functions

```
//create object without class
//The function distance does not care whether the arguments are an instance of
the class Point

function distance(p1, p2) {
    return Math.sqrt((p1.x - p2.x) ** 2 + (p1.y - p2.y) ** 2)
    //Exponentiation operator (**)

}

console.log(distance({ x: 1, y: 1 }, { x: 2, y: 2 }))
//1.4142135623730951
```

How to Compare Objects in JavaScript

1. Referential equality: ==, ===
2. Manual comparison of properties values.
3. Shallow Equality check the properties values for equality.

1. Referential equality

- Both are the same object means both object point to the same object instances.
- Compare objects:
  - The strict equality operator ===
  - The loose equality operator ==

```
//Object Comparing
let student = { id: 1, name: "Joe" }
let newStudent = { id: 2, name: "Joe" }
let oldStudent = { id: 1, name: "Joe" }
let alumniStudent = student;
```

```
if (student == alumniStudent) { //true
  console.log("student equals to alumni student by ==")
  //student equals to alumni student by ==
}
if (student == newStudent) { //false
  console.log("student equals alumni student by ==")
}
if (student === alumniStudent) { //true
  console.log("student strictly equals to alumni student")
  //student strictly equals to alumni student by ===.
}

if (student === newStudent) { //false
  console.log("student strictly equals to new student by ===")
}
```

2. Manual Comparison

- A manual comparison of properties values.

```
//compare properties manually
function isStudentEqual(object1, object2) {
  return object1.id === object2.id
}
```

```

console.log(isStudentEqual(student, oldStudent)) //true
console.log(isStudentEqual(student, alumniStudent))//true
```

3. Shallow Equality

The `Object.keys()` method returns an array of a given object's own enumerable property **names**, iterated in the same order that a normal loop would.

```
//Shallow Equality
let book1 = {
  isbn: 123456789,
  title: "JavaScript",
}

let book2 = {
  isbn: 123456789,
  title: "JavaScript",
}
```

```
function shallowEquality(object1, object2){
  const keys1=Object.keys(object1)
  const keys2=Object.keys(object2)

  if(keys1.length !== keys2.length){
    return false
  }
  for(let key of keys1){
    if(object1[key] !== object2[key] ){
      return false
    }

  }
  return true
}
```

```
console.log("shallow equality: " + shallowEquality(book1, book2)) //true
```

Object Prototypes

- **Prototypes** are the mechanism by which JavaScript objects inherit features from one another.
- **JavaScript** is often described as a prototype-based language — to provide inheritance, objects can have a prototype object, which acts as a template object that it inherits methods and properties from.

Prototype Chaining

- ECMA-262 describes **prototype chaining** as the primary method of **inheritance** **in** ECMAScript.
- The object created by **new** Object() or **object literal** inherit from `Object.prototype`
- Similarly, the object created by `new Array()` uses `Array.prototype` as its prototype, and the object created by `new Date()` uses `Date.prototype` as itsprotot ype.
- `Date.prototype` inherits properties from `Object.prototype`, so a `Date` object created by `new Date()` inherits properties from both `Date.prototype` and `Object.prototype`.
- This linked series of prototype objects is known as a ***prototype chain.***

Prototype Chaining

- JavaScript objects have a set of **own properties** and they also inherit a set of properties from their prototype object.

```
let o = {}                 // o inherits object methods from Object.prototype
o.x = 1                    // and it now has an own property x.
let p = Object.create(o)   // p inherits properties from o and Object.prototype
p.y = 2                    // and has an own property y.
let q = Object.create(p)   // q inherits properties from p, o, and Object.prototype
q.z = 3                    // and has an own property z.
let f = q.toString()       // toString is inherited from Object.prototype
q.x + q.y                  // 3; x and y are inherited from o and p
```

Prototype Chaining

`prototype.isPrototypeOf(object)`

object - the object whose prototype chain will be searched.
Return a Boolean indicating whether the calling object lies in the prototype chain of the specified object.

```
//define our own class and
//constructor functions
class Rectangle{
  constructor(width, height){
    this.width=width
    this.height=height
  }
  area(){
    return this.width*this.height
  }
}
let rec1=new Rectangle (2, 3)
console.log(rec1.area()) //6
```

```
//create object with Object.create()
let square = Object.create(rec1)
square.perimeter = function() {
  return 4 * this.width
}
console.log(square.width) //2
console.log(square.height)//3
console.log(square.area())//6
console.log(square.perimeter())//8
console.log(Object.prototype.isPrototypeOf(rec1))//true


console.log(Rectangle.prototype.isPrototypeOf(square))//true
console.log(Object.prototype.isPrototypeOf(square))//true
```

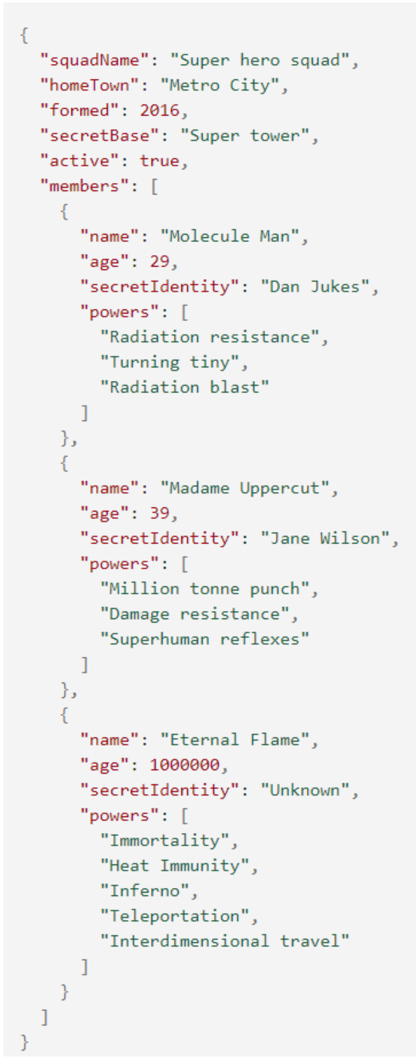
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/isPrototypeOf

JSON – JavaScript Object Notation

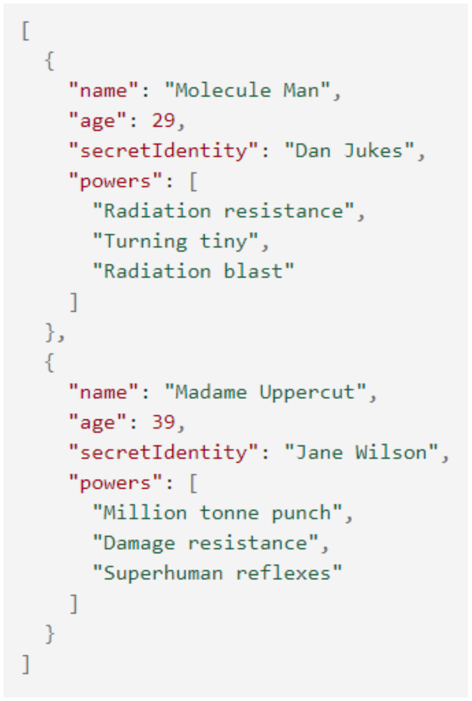
- JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax.
- It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).
- Even though it closely resembles JavaScript object literal syntax, it can be used independently from JavaScript, and many programming environments feature the ability to read (parse) and generate JSON.
- A JSON string can be stored in its own file, which is basically just a text file with an extension of .json, and a MIME type of application/json.

JSON structure

- **JSON is a string** whose format very much resembles JavaScript object literal format.
- JSON **requires double quotes** to be used around strings and property names. **Single quotes are not valid** other than surrounding the entire JSON string.
- You can include the same basic data types inside JSON as you can in a standard JavaScript object — strings, numbers, arrays, booleans, and other object literals.
- JSON is purely a string with a specified data format — **it contains only properties, no methods**.
- We can also convert arrays to/from JSON.



```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```



```
[
  {
    "name": "Molecule Man",
    "age": 29,
    "secretIdentity": "Dan Jukes",
    "powers": [
      "Radiation resistance",
      "Turning tiny",
      "Radiation blast"
    ]
  },
  {
    "name": "Madame Uppercut",
    "age": 39,
    "secretIdentity": "Jane Wilson",
    "powers": [
      "Million tonne punch",
      "Damage resistance",
      "Superhuman reflexes"
    ]
  }
]
```

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

JSON.stringify()

- The `JSON.stringify()` method converts a JavaScript object or value to a JSON string.

```
console.log(JSON.stringify({ x: 5, y: 6 }))
// expected output: "{"x":5,"y":6}"
```

Ways to Check If an Object Is Empty

```
const emptyObj = {}
```

```
//way#1 - using JSON.stringify()
if (JSON.stringify(emptyObj) === '{}')
  console.log('1. emptyObj is empty object')
```

```
//way#2 – using Object.keys()

if (Object.keys(emptyObj).length === 0)
  console.log('2. emptyObj is empty object')
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys

Spread (...) in object literals

- In an object literal, the spread syntax enumerates the properties of an object and adds the key-value pairs to the object being created.

```
const obj1 = { foo: 'bar', x: 42 }
const obj2 = { foo: 'baz', y: 13 }

const clonedObj = { ...obj1 }
// Object { foo: "bar", x: 42 }
const clonedWithReplace = { ...obj1, foo: 'abc' }
// Object { foo: "abc", x: 42 }
const mergedObj = { ...obj1, ...obj2 }
// Object { foo: "baz", x: 42, y: 13 }
```

Note that the property value of obj2 will replace the property value of obj1 in the merged object.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax#spread_in_object_literals

Object Destructuring

```
const student = {
  id: 1001,
  fullname: 'Somchai Jaidee',
  email: 'somchai@example.com'
}

let id = student.id
let fullname = student.fullname
let email = student.email

console.log(id) //1001
console.log(fullname) //Somchai Jaidee
console.log(email) //Somchai@example.com
```

destructuring →

```
const student = {
  id: 1001,
  fullname: 'Somchai Jaidee',
  email: 'somchai@example.com'
}

let { id, fullname, email } = student

console.log(id) //1001
console.log(fullname) //Somchai Jaidee
console.log(email) //Somchai@example.com
```

Object Destructuring

- The **destructuring assignment** syntax is a JavaScript expression that makes it possible to **unpack values from arrays, or properties from objects, into distinct variables.**

```
let a, b, rest;
[a, b] = [5, 10]

console.log(a)//5
console.log(b)//10

[a, b, ...rest] = [5, 10, 15, 20, 25]
console.log(rest) // [15,20,25]
```

```
({ a, b } = { a: 10, b: 20 });
console.log(a) // 10
console.log(b) // 20

({ a, b, ...rest } = { a: 10, b: 20, c: 30, d: 40 })
console.log(a) // 10
console.log(b) // 20
console.log(rest) // {c: 30, d: 40}
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

Nested Object Destructuring

```
const msg = {
  sender: 'Somsak',
  recipient: 'Pornsuda',
  content: {
    header: 'Reminder our party',
    body: 'let see you in the party'
  }
}

const {content: { header }} = msg

console.log(header)//Reminder our party
```