Hyperiondev

TASK

# Programming in JavaScript III: JSON

Visit our website

# Introduction

## WELCOME TO THE JSON TASK!

You will find that JavaScript objects and arrays are used when creating many web applications! These are essential data structures that are used very often in web development. The fact that these data structures need to be used with HTTP when doing web development means that we need to use two very important tools: the Web Storage API and JSON. Both of these tools are introduced in this task. You will also be introduced to the concept of object-oriented programming and how to create JavaScript objects. This is an extremely important concept! JavaScript uses objects extensively. By the end of this task, you will see how you have actually been using JavaScript objects all along!

Get in touch

**Connect for support**

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our expert code reviewers are happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

## RECAP: OBJECT-ORIENTED PROGRAMMING: WHAT IS AN OBJECT?

Before we start working with JSON, you need to be comfortable with the concept of objects. So, what is an object? To answer this question, let's look at a typical object we might find on a webpage: a button. By its very nature, a button should make something happen when you press it — it should perform an *action*. A button also needs to have *attributes*: it needs to be a certain size and shape, and it needs a colour and possibly a label or image on it to give an indication of what pressing it might do. These characteristics are the foundation of any object you create in programming: an object needs to have *attributes* and it needs to have *methods* to enable it to perform a certain action. Have a look at the button below. What are its attributes? What action does it execute?


Simply put, Object-Oriented Programming (OOP) is a way of creating neat packages of code (i.e. objects) that have certain attributes and can perform specific actions. This is a paradigm shift away from the procedural paradigm that is described under the next subheading.

## PROCEDURAL PROGRAMMING VS OBJECT-ORIENTED PROGRAMMING

Procedural code executes in a waterfall fashion. We proceed from one statement to another in sequence. The data are separated from the code that uses them. Functions are defined as standalone modules, but they have access to variables we've declared outside the function. These two properties: sequential execution of code and the separation of data and the code that manipulates the data are what distinguish procedural code.

In contrast, with object-oriented programming, a system is designed in terms of objects that communicate with each other to accomplish a given task. Instead of separating data and code that manipulate the data, these two are encapsulated into a single module. Data is passed from one module to the next using methods.

## JAVASCRIPT OBJECTS

JavaScript is an object-based programming language. An understanding of objects will enable you to understand JavaScript better. Almost everything in JavaScript is an object!

OOP serves to allow the components of your code to be as modular as possible. The benefits of this approach are a shorter development time and easier debugging because you're reusing program code that has already been proven.

**Take note:**

JavaScript is not strictly an object-oriented programming language. It is an object-based scripting language. Like pure object-oriented languages, JavaScript works with objects, but with JavaScript, objects are created using prototypes instead of classes. A prototype is basically a constructor function. You will learn more about this distinction later.

**ES6 update:** ES6 allows us to create objects using keywords (like class, super, etc.) that are used in class-based languages although, under the hood, ES6 still uses functions and prototypal inheritance to implement objects. For an example of an ES6 class, see **here**.

We've covered the basic theory of what objects are and why we use them, now let's get coding and learn to create JavaScript objects.

## CREATING DEFINED OBJECTS

There is more than one way of creating objects. We will consider two key ways in this task.

### Method 1: Using Object Literal (easiest)

Let's consider the following object declaration:

```
let car = {
        brand:"Porsche",
        model:"GT3",
        year:2004,
        colour:"White",
        howOld:function(){getFullYear() - this.year;}
};
```

Here an object is declared with four different properties, which is then stored in an object variable called `car`. This object also contains a method called `howOld`.

- *Object Properties/Attributes:* As you know, an object comprises a collection of named values, called attributes. To demonstrate this concept, we'll consider the car object:

| Property | Value |
|---|---|
| Brand | Porsche |
| Model | GT3 |
| Year | 2004 |
| Colour | White |

- *Object Methods:* Methods (functions) are a series of actions that can be carried out on an object. An object can have a primitive value, a function, or even other objects as its properties. Therefore, an object method is simply a property that has a function as its value. Notice that the `car` object above contains a method called `howOld`. This object method will calculate how old the car is by checking the current year and subtracting the production year of the car. You'll notice `this.year` was used instead of `car.year`. `this` is a keyword that you can use within a method to refer to the *current object*. `this` is used because it's a property within the same object, thus it simply looks for a `year` property within the object and uses that value.

This first method for creating objects that we have considered is used to create a single object at a time. What if you want to create a number of objects that all have the same properties and methods but different values? The second method that you will consider next provides an effective way of creating many objects using a single object constructor.

**Method 2: Using Object Constructor**

The second way of creating an object is by using an object constructor. A *constructor* is a special type of function that is used to make or construct a number of different objects.

Consider the example below. The function `carDescription` is the constructor. It is used to create 3 different car objects: car1, car2 and car3.

```
function carDescription(brand, model, year, colour) {
   this.brand = brand;
   this.model = model;
   this.year = year;
   this.colour = colour;
};

let car1 = new carDescription("Porsche", "GT3", 2012, "White");
let car2 = new carDescription("Ford", "Fiesta", 2015, "Red");
let car3 = new carDescription("Opel", "Corsa", 2014, "White");
```

The dot or bracket notation can be used to modify any value in an object. For example, if you had your Porsche spray painted black, you could change the colour property of your car as shown below:

```
car1.colour = "Black";
```

To delete properties of an object, you use the delete keyword:

```
delete car2.colour;
```

**Take note:**

You may have noticed the use of the **this** keyword in the code above. In JavaScript, **this** is used to refer to the object itself. So, in the example above, **this** is used to refer to **carDescription** to identify the brand, model, year and colour of each new object. This is what enables us to say **car1.brand = "Porsche"** — because **this.brand** shows that we are referring to this specified object.

## SOME OBJECTS THAT YOU HAVE ALREADY ENCOUNTERED

As noted previously, JavaScript is an object-based language. Most of the built-in code that you have worked with so far has, therefore, used objects. For example, whenever a web page is loaded an object called *document* is created. This object contains methods and properties that describe and can be used to manipulate the webpage's structure and content.

When you use code such as:

```
let htmlSelect = document.getElementById('personList');
```

You have been using the *document* object. Similarly, every time you create an array, you are actually creating an array object that is defined by an array class. The whole JavaScript programming language is built using objects.

## FUNCTION MEETS OBJECT

Now that you have a good understanding of functions and objects, let's consider the declaration of an object with implementation through a function (after all, you do want the object to serve a purpose):

```
let loaded = {};
loaded.testing = function(signal) {
       alert("Hello World! " + signal);
       loaded.signal = signal;
}
loaded.testing("This page has loaded!");
```

Here a blank object is created. An object method is created with the intention of displaying a message to the user when the page has loaded. The function is called with a particular value which is then set as the value of the `loaded.signal` property.

## END OF RECAP

## SENDING OBJECTS BETWEEN A WEB SERVER AND CLIENTS

HTTP (HyperText Transfer Protocol) is the protocol that allows for information to be transferred across the internet. Everything that is transferred over the web is transferred using HTTP. There are two very important facts about HTTP that we need to keep in mind, though:

1. **HTTP is a stateless protocol.** That means HTTP doesn't see any link between two requests being successively carried out on the same connection. Cookies and the Web Storage API are used to store necessary state information.

2. **HTTP transfers text (not objects or other complex data structures).** JSON converts data structures, like objects, into text that can be transferred using HTTP.

## THE WEB STORAGE API: SESSION STORAGE

Thus far, we have used variables to store data that is used in our programs. When storing data that is used for web applications it is important to keep in mind that HTTP is a *stateless protocol*. That basically means that the web server doesn't store information about each user's interaction with the website. For example, if 100 people are shopping online, the webserver that hosts the online shopping application doesn't necessarily store the state of each person's shopping experience (e.g. how many items each person has added or removed from their shopping cart). Instead, that type of information is stored on the browser using **Cookies** or the *Web Storage API* (the more modern and efficient solution for client storage). The Web Storage API stores data using key-value pairs. This mechanism of storing data has, to a large extent, replaced the use of cookies.

The Web Storage API allows us to store state information in two ways,:

1. *sessionStorage* stores state information for each given origin for as long as the browser is open.

2. *localStorage* stores state information for each given origin even when the browser is closed and reopened.

You can add items to sessionStorage as shown below:

```
sessionStorage.setItem("totalPersonObjs", 1);
```

This will add the key value pair {"totalPersonObjs", 1} to sessionStorage. You could retrieve a value from sessionStorage as shown below:

```
let total = parseInt(sessionStorage.getItem("totalPersonObjs"));
```

For more information about how to use sessionStorage see “personObjectEG.js” **here**.

## JAVASCRIPT OBJECT NOTATION (JSON)

As stated previously, everything that is transferred over the web is transferred using HTTP. As the name suggests, this protocol can transfer *text*. All data that is transferred across the web is, therefore, transferred as text. As such, we cannot transfer JavaScript objects between a web server and a client. *XML* and *JSON* are commonly used to convert JavaScript objects into a format that can be transferred with HTTP.

### What is XML?

eXtensible Markup Language (XML) is a markup language that is used to annotate text or add additional information. Tags are used to annotate data. These tags are not shown to the end-user, but are needed by the ‘machine’ to read and subsequently process the text correctly.

Below is an example of XML. Notice the tags. They are placed on the left and the right of the data you want to markup, so as to wrap around the data.

```
<book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications
      with XML.</description>
</book>
```

This is the general pattern that we have to follow for all tags in XML.

```
<opening tag>Some text here.</closing tag>
```

Looking at the example of XML above may remind you of HTML. They are both markup languages but, whereas HTML focuses on *displaying* data, XML just *carries data*.

XML files don’t do anything except carrying information wrapped in tags. We need software to read and display this data in a meaningful way.

XML is used to structure, store and transport data independent of hardware and/or software.

**What is JSON?**

JSON, or JavaScript Object Notation, is a syntax for converting objects, arrays, numbers, strings and booleans into a format that can be transferred between the web server and the client. Like XML, JSON is language-independent.

Only text data can be exchanged between a browser and a server. JSON is text and any JavaScript object can be converted into JSON, which can then be sent to the server. Any JSON data received from the server can also be converted into JavaScript objects. We can, therefore, easily work with data as JavaScript objects, without any complicated parsing and translations. Data also has to be in a certain format in order to store it. JSON makes it possible to store JavaScript objects as text which is always one of the legal formats.

JSON is much more lightweight than XML. It is shorter, and quicker to read and write. JSON also doesn't use end tags and can use arrays. XML also has to be parsed with an XML parser while JSON can be parsed by a standard JavaScript function.

**JSON Syntax**

The JSON syntax is a subset of the JavaScript syntax. However, this does not mean that JSON cannot be used with other languages. JSON works well with PHP, Perl, Python, Ruby, Java, and Ajax, to name but a few.

Below are some of the JSON Syntax rules:

- Data are in key/value pairs
- Property names must be double-quoted strings
- Data are separated by commas
- Objects are held by curly braces - { }
- Arrays are held by square brackets - [ ]

As mentioned above, JSON data are written as key/value pairs. Each pair consists of a field name or key, which is written in double quotes, a colon and a value.

Example:

```
"name" : "Jason"
```

The key must be a string, enclosed in double quotes, while the value can be a string, a number, a JSON object, an array, a boolean or null.

JSON uses JavaScript syntax, so very little extra software is needed to work with JSON within JavaScript. The file type for JSON files is **.json** and the MIME type for JSON text is "application/json".

**JSON objects**

Below is an example of a JSON object:

```
let myObj = { "name":"Jason", "age":30, "car":null };
let x = myObj.name;
```

The key must be a string, enclosed in double quotes, while the value can be a string, a number, a JSON object, an array, a boolean or null.

Object values can be accessed using the dot (**.**) notation (as in the example above). Object values can also be accessed using square brackets (**[]**) notation:

Example:

```
let myObj = { "name":"Jason", "age":30, "car":null };
let x = myObj["name"];
```

The dot or bracket notation can be used to modify any value in a JSON object:

```
myObj.age = 31;
```

To delete properties from a JSON object, you use the delete keyword:

```
delete myObj.age;
```

It is also possible to save an array of objects with JSON. See the example below where an array of objects is stored. The first object in the array describes a person object with the name “Tom Smith” and the second object describes a person called “Jack Daniels”:

```
let arrayofPersonObjects = [{ "name": { "first": "Tom", "last": "Smith" },
"age": "21", "gender": "male", "interests": "Programming" },
```

```
    { "name": { "first": "Jack", "last": "Daniels" }, "age": "19", "gender":
"male", "interests": "Gaming" }
];
```

You can use the *for-in* loop to loop through an object's properties:

```
let myObj = { "name":"Jason", "age":30, "car":null };
for (x in myObj) {
    document.getElementById("demo").innerHTML += x;
}
```

To access the property values in a *for-in* loop, use the bracket notation:

```
let myObj = { "name":"Jason", "age":30, "car":null };
for (x in myObj) {
      document.getElementById("demo").innerHTML += myObj[x];
}
```

A JSON object can contain other JSON objects:

```
let myObj = {
    "name":"Jason",
    "age":30,
    "cars": {
        "car1":"Ford",
        "car2":"BMW",
        "car3":"VW"
    }
 }
```

To access a nested JSON object, simply use the dot or bracket notation:

```
let x = myObj.cars.car2;
//or
let x = myObj.cars["car2"];
```

**JSON methods**

As mentioned before, one of the key reasons for using JSON is to convert certain JavaScript data types (like arrays) into text that can be transferred using HTTP. To be able to do this we use the following two JSON methods:

- `JSON.parse()`

By parsing the data using **`JSON.parse()`**, the data becomes a JavaScript object. Imagine that you receive the following text from a web server:

```
'{ "name":"Jason", "age":30, "city":"New York"}'
```

Using the **`JSON.parse()`** function, the text is converted into a JavaScript object:

```
let obj = JSON.parse('{ "name":"Jason", "age":30, "city":"New York"}');
```

You can then use the JavaScript object in your page, as follows:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;
</script>
```

- **`JSON.stringify()`**

  All data you send to a web server has to be a string. To convert a JavaScript object into a string, you use **`JSON.stringify()`**.

  Imagine that you have the following JavaScript object:

```
let obj = { "name":"Jason", "age":30, "city":"New York"};
```

  Using the **`JSON.stringify()`** function converts this object into a string:

```
let myJSON = JSON.stringify(obj);
```

  **`myJSON`** is now a string, and can be sent to a server:

```
let obj = { "name":"Jason", "age":30, "city":"New York"};
let myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
```

# Instructions

Open the **Examples folder** in Visual Studio Code and read through the comments before attempting these tasks.

Getting to grips with JavaScript takes practice. You will make mistakes in this task. This is completely to be expected as you learn the keywords and syntax rules of this programming language. It is vital that you learn to debug your code. To help with this remember that you can:

- Use either the JavaScript console or Visual Studio Code (or another editor of your choice) to execute and debug JavaScript in the next few tasks.
- Remember that if you really get stuck, you can contact an expert code reviewer for help.

# Compulsory Task 1

Follow these steps:

- Create a webpage that can be used to let a user store information about a catalogue of books.
- Make use of session storage to store the values.
    - The user should be able to add information (e.g. author, title, genre, reviews, etc.) about their favourite books.
    - All the information about all the books added by the user should be listed on the webpage.
    - The user should also be able to remove or edit information for a book.

# Compulsory Task 2

Follow these steps:

- Create a basic HTML file. You are required to create a budgeting website with the following specifications:
- Make use of session storage to store the values.
- Create an *income* object where you can put in the following information as attributes:
    - Name, as a string (E.g. Salary)
    - The amount, as a number (E.g. 4000)
    - Whether or not it is recurring, as a boolean
- Create 5 different objects to represent income from different places.
- Create an *expenses* object where you can put in the following information as attributes:
    - Name, as a string (E.g. Groceries)
    - The amount, as a number (E.g. 350)
    - Whether or not it is recurring, as a boolean
- Create 5 different objects to represent different expenses.
- Using a prompt box, display the income items and let the user add another entry.
- Using a prompt box, display the expenses items and let the user add another entry.
- Display the total amount of disposable income (income minus expenses)
- Using a prompt box, ask the user how much of their disposable income they would like to put into savings.
- Finally, create an alert to display the total amount of disposable income left.

Rate us

# Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

**Click here** to share your thoughts anonymously.