



TASK

JSON and Client-Server Communication

Visit our website

Introduction

WELCOME TO THE JSON AND CLIENT-SERVER COMMUNICATION 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 we need a way to transfer these objects; for that, we will look into JSON. Given the emphasis on objects in this task, you should revisit concepts from Object-Oriented Programming. By the end of this task, you will see how you have actually been using JavaScript objects all along!

Before discussing transferring JavaScript objects in detail, let's first look into user input and output and DOM manipulation. These concepts will be used while working with JavaScript to create dynamic web pages and respond to user interactions.

USER INPUT AND OUTPUT WITH JAVASCRIPT

JavaScript enables developers to engage with users by receiving input and delivering output through interactive dialogue boxes. Dialogue boxes are a user interface element, built into web browsers, that interact directly with the user while they are interacting with a web page.

These dialogue boxes are known as a modal window. A modal window interrupts the normal execution of code, disables user interaction from the main application page, and requires users to focus on a specific window before continuing. A user can dismiss the dialogue box by clicking the "OK" button that's typically present, after which the execution of the code resumes.

It's important to note that these dialogue boxes are not part of the development environment itself, such as VS Code, where the code is written and executed.

Two fundamental functions for engaging with these dialogue boxes are **`prompt()`** and **`alert()`**.

Capturing user input with **`prompt()`**

In JavaScript, the **`prompt()`** function triggers a pop-up dialogue that prompts users to input data such as text or numbers. The prompt function accepts a message as its parameter and then displays it as a prompt to the user. After the user responds, the function returns the *entered value* that can be stored in a variable for later use.

Displaying user output with alert()

In JavaScript, the **alert()** function triggers a pop-up dialogue box that alerts the user of an action or event by showcasing a specified message. Similar to the prompt function, the alert function takes a message as its parameter, and upon execution, displays the message to the user within the pop-up dialogue box.

Combining input and output

You can use both **prompt()** and **alert()** functions together to create interactive experiences. Let's take a look at the following example that takes a user's input and immediately provides a response using an alert:

```
// prompt user to input data
const userName = prompt("Please enter your name:");

// display a greeting using the alert function if the data is valid
if (userName !== null && userName !== "") {
    const greeting = "Hello, " + userName + "!";
    alert(greeting);
}
// Display an appropriate response if the user dismisses the dialogue box
// without entering valid data
else {
    alert("You did not provide a valid name.");
}
```

Try this:

- Open a new window in your browser and hit F12 to inspect it.
- In the console tab, enter **alert("This is an alert!")** and hit enter.
- As you execute the command, you will notice a dialogue box that promptly appears on your screen. This is the result of the **alert()** function in action.

DOCUMENT OBJECT MODEL (DOM)

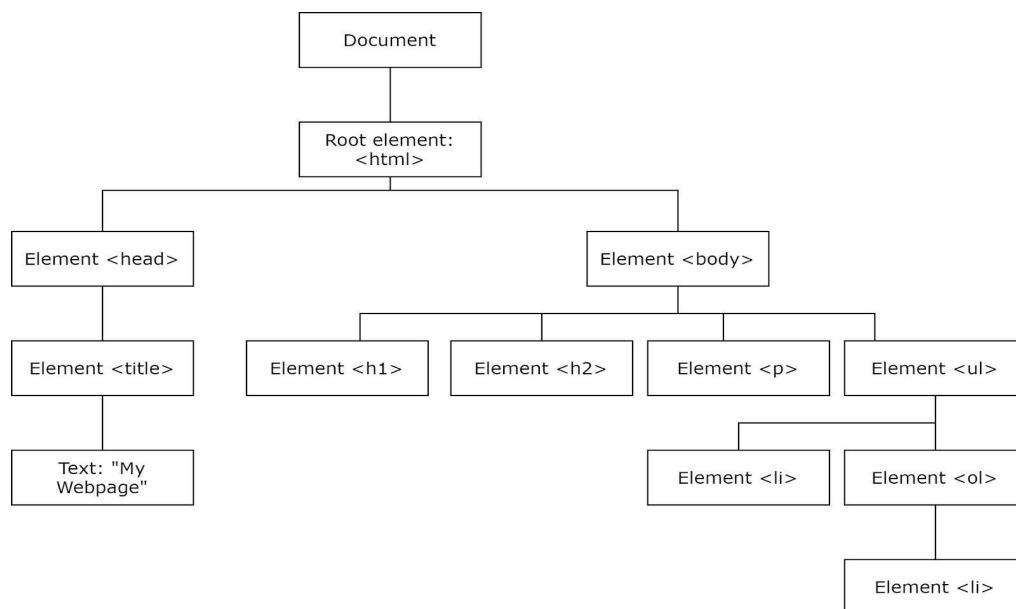
When a web page is loaded, the browser creates a Document Object Model of the page. DOM stands for **D**ocument **O**bject **M**odel and is a W3C (World Wide Web Consortium) standard for how to get, change, add, or delete HTML elements.

It allows the user to interact with the structure, content, and style of HTML and XML documents. DOM represents the document as a tree-like structure where each element, attribute, and content is represented as an object. This helps developers dynamically modify and interact with web pages using programming languages like JavaScript. The top-level object is the document object, which represents the

entire web page. Let's look at a skeleton HTML code and understand how it correlates to DOM.

```
<!DOCTYPE html>
<html>
<head>
  <title>My Webpage</title>
</head>
<body>
  <h1> </h1>
  <h2> </h2>
  <p> </p>
  <ul>
    <li> </li>
    <ol>
      <li> </li>
    </ol>
  </ul>
</body>
</html>
```

Below is the corresponding DOM tree for the above HTML skeleton code. We can use this object model to create dynamic pages by manipulating this tree, i.e., the DOM.



Now that we know what the DOM data structure is, let's understand how we can access and modify elements using it. DOM *methods* are used to perform actions on HTML elements, such as getting, adding, or deleting elements, while DOM *properties* are used to change the values of HTML elements.

Getting elements using DOM methods

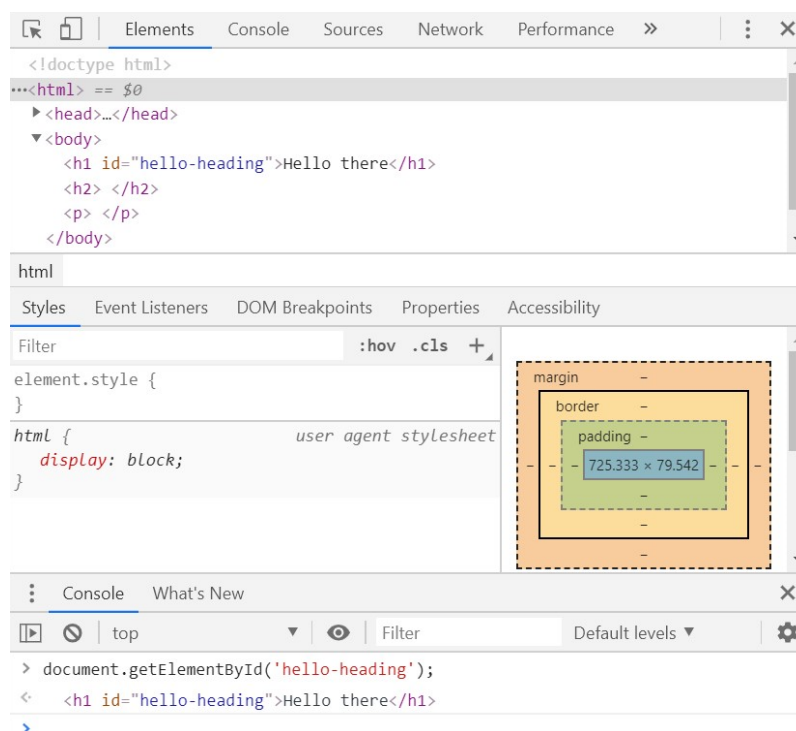
To access any element in an HTML page, you always start by accessing the document object with the `'document.'` syntax.

The `document.getElementById('id')` is a core DOM method in JavaScript. It retrieves a specific element from the DOM based on its unique ID attribute. The parameter `'id'` is a string value of the `'id'` attribute of the HTML element which the user wants to retrieve. If an element with the specified ID exists in the DOM, then the method returns the reference of that element as an object. If an element with that ID does not exist, then `'null'` is returned.

For example, if you have an HTML element defined as `<h1 id="hello-heading">Hello there</h1>`, you can access this element in JavaScript using its `id` attribute:

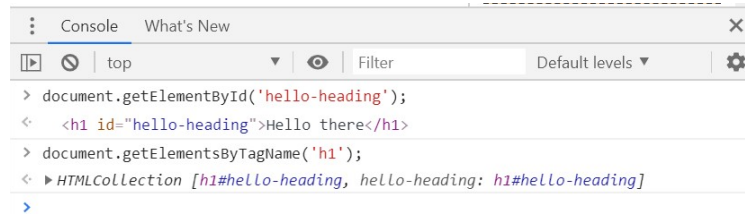
```
var headingElement = document.getElementById('hello-heading');
```

If we opened up this HTML file in Chrome and inspected it, as seen in the image below, we could get the `h1` element by its ID by typing in `document.getElementById('hello-heading');`



As you can see in the console at the bottom, the **h1** element has been returned. For an in-depth overview, you can learn more about the `getElementById()` [here](#).

We can also achieve this by using the `document.getElementsByTagName()` method:

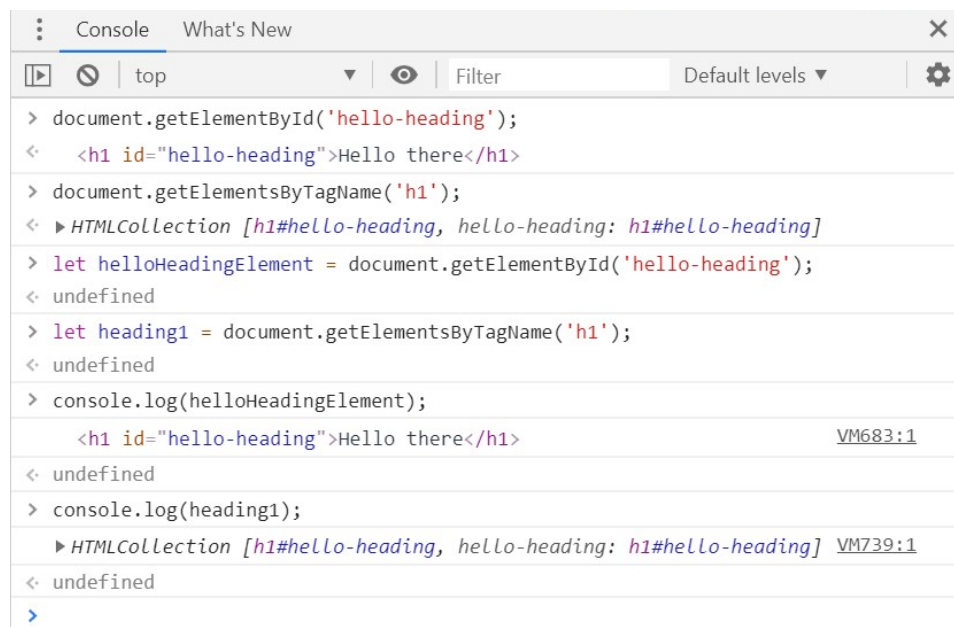


```
> document.getElementById('hello-heading');
< <h1 id="hello-heading">Hello there</h1>

> document.getElementsByTagName('h1');
▶ HTMLCollection [h1#hello-heading, hello-heading: h1#hello-heading]
```

This returns the HTML Collection of all occurrences of that particular **h1** element. In this case, only one element is present and is returned. Although this collection may look like an array, in square brackets, it behaves slightly differently. These can be assigned to variables to make use of when we make changes to these elements. This can simply be achieved by making the `getElement` method the value of the variable.

In the example below, we have assigned the results of both methods to variables and printed them to the console. Note that while the examples above have been in the console, when making web pages more dynamic with JavaScript the code will be in a JavaScript file that the HTML file will import:



```
> document.getElementById('hello-heading');
< <h1 id="hello-heading">Hello there</h1>

> document.getElementsByTagName('h1');
< ▶ HTMLCollection [h1#hello-heading, hello-heading: h1#hello-heading]

> let helloHeadingElement = document.getElementById('hello-heading');
< undefined

> let heading1 = document.getElementsByTagName('h1');
< undefined

> console.log(helloHeadingElement);
    <h1 id="hello-heading">Hello there</h1> VM683:1
< undefined

> console.log(heading1);
    ▶ HTMLCollection [h1#hello-heading, hello-heading: h1#hello-heading] VM739:1
< undefined

>
```

Changing elements using DOM properties

One of the fundamental tasks in DOM manipulation is altering the content of HTML elements. The most commonly used property for this purpose is `.innerHTML`. This property allows you to retrieve or modify the HTML content within an element,

encompassing not only text but also child elements, like `<html>` and `<body>`. Looking at our first example above, we could change the `h1` element, with the ID `'hello-heading'`, by doing the following:

```
document.getElementById('hello-heading').innerHTML = 'Hello World!';
```

It's worth noting that besides `.innerHTML`, the DOM offers various other methods for element manipulation. For instance, there's also `.outerHTML`, which allows you to modify an entire element along with its container, such as a `<div>` block.

If you're interested in exploring more techniques for getting and changing elements, please read this [article](#). With these tools at your disposal, you can dynamically alter your web page's content to create interactive and engaging user experiences.



Take note:

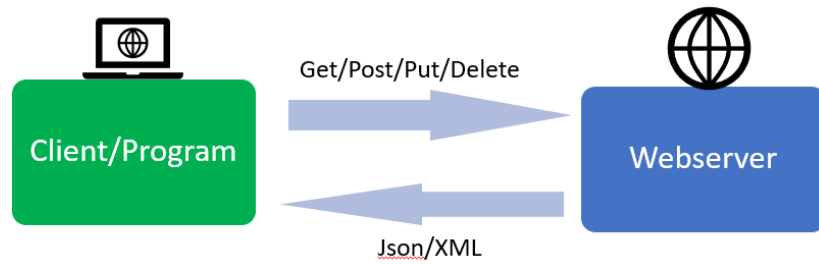
For more in-depth information on the DOM, review the [Document Object Model \(DOM\)](#) page on the Mozilla Developer Network.

SENDING OBJECTS BETWEEN A WEB SERVER AND CLIENTS

Everything that is transferred over the web is transferred using HTTP (HyperText Transfer Protocol). 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.

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).** XML and JSON are commonly used to convert data structures, like objects, into a text format that can be transferred using HTTP.



EXTENSIBLE MARKUP LANGUAGE (XML)

e**X**tensible **M**arkup **L**anguage (XML) is a markup language 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 process the text correctly.

Below is an example of XML. Notice that the tags are placed on the left and right of the data you want to markup 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 do not do anything except carry informational data wrapped in tags. XML structures, stores, and transports the data independent of hardware or software. We do, however, need software such as XML parsers or applications capable of interpreting XML data, like web browsers, text editors with XML support, or specific development tools designed to work with XML to read and display this data in a meaningful way.

JAVASCRIPT OBJECT NOTATION (JSON)

JSON, or **J**ava**S**cript **O**bject **N**otation, 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. JSON is language-independent like XML.

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 work more efficiently with data as JavaScript objects without the need for complicated parsing and translations. Data must also be in a certain format 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 as it is shorter and quicker to read and write. JSON also does not use end tags. XML must also be parsed with an XML parser, while a standard JavaScript function can parse JSON.

JSON Syntax

The JSON syntax is a subset of the JavaScript syntax. However, this does not mean 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 JSON Syntax rules:

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

As mentioned above, JSON data is written as key/value pairs. Each pair consists of a field name or key 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 you don't need any additional JS library to work with JSON within JavaScript. The file type for JSON files is **.json** and the MIME (Multipurpose Internet Mail Extensions) type for JSON text is "**application/json**". MIME is a standard that extends the format of email messages to support text in character sets other than ASCII. It also supports audio messages, videos, and images. In the context of an email, MIME headers are used to specify the type of content within an email. However, MIME is not just limited to emails. It is also used in HTTP to specify the type of the data being sent in an HTTP response i.e audios, videos, and images.

JSON objects

Let's take a look at the example below of a JSON object. Object values can be accessed using either the dot (.) notation or the square brackets ([]) notation.

```
let myObj = { "name":"Jason", "age":30, "car": null };
// assign 'name' to x using dot notation
let x = myObj.name;
// or assign 'name' to x using bracket notation
let x = myObj["name"];
```

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

```
myObj.age = 31; // using dot notation
myObj["age"] = 31; // using bracket notation
```

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

```
delete myObj.age;
```

Saving an array of objects with JSON is also possible. 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 a for-in loop together with bracket notation to loop through an object's properties and access the property values, as shown below. [hasOwnProperty\(\)](#) is a built-in method in JavaScript that determines whether an object has a specific property as its own and is not inherited from an object's prototype:

```
var myObj = { "name":"Jason", "age":30, "car":null };
for (var key in myObj) {
  if (myObj.hasOwnProperty(key)) {
    console.log(key + ": " + myObj[key]);
  }
}
```

Once the above has been logged to the console, you can then access the property values in a for-in loop, using the bracket notation, to load and display it in the browser:

```
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 do this, we use the following two JSON methods:

- **JSON.parse()**

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

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

By 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 now access the properties of this object using the dot notation, such as **obj.name**, **obj.age**, and **obj.city**.

Once we have the data as a JavaScript object, we can use it to manipulate the content of our HTML page. In the following example, we will display the name and age properties of our object inside a paragraph element with the id **"demo"**:

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

When this code is run, the content of the paragraph element will be updated to show:

```
Jason, 30
```

Here is the complete code

```
<!DOCTYPE html>
<html>
<head>
  <title>JSON Parsing Example</title>
</head>
<body>
  <p id="demo"></p>

  <script>
    // JSON string received from the server
    var jsonString = '{ "name":"Jason", "age":30, "city":"New York"}';

    // Parsing the JSON string into a JavaScript object
    var obj = JSON.parse(jsonString);

    // Accessing the name and age properties of the object and
    inserting them into the paragraph element with id "demo"
    document.getElementById("demo").innerHTML = obj.name + ", " +
obj.age;
  </script>
</body>
</html>
```

- **JSON.stringify()**

All the 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);
```

Now, **myJSON** is a string that represents the original object, and it can be sent to a server or displayed within an HTML document. Here's how you can display the JSON string inside a paragraph element with the id "demo":

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

Here is the complete code.

THE

```
<!DOCTYPE html>
<html>
<head>
  <title>JSON Example</title>
</head>
<body>
  <p id="demo"></p>
  <script>
    let obj = { "name": "Jason", "age": 30, "city": "New York" };
    let myJSON = JSON.stringify(obj);
    document.getElementById("demo").innerHTML = myJSON;
  </script>
</body>
</html>
```

WEB STORAGE API: SESSION STORAGE

Thus far, we have used variables to store data that is used in our programmes. 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 web server 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](#).

CONCLUSION

JSON plays a key role in web development, making data transfer and storage straightforward. Its simplicity and compatibility with JavaScript ensure it remains a top choice for developers. As you move forward, keep JSON in mind to enhance your web applications.

Instructions

Read and run the accompanying examples files provided before doing the compulsory task to become more comfortable with the concepts covered in this task.

Compulsory Task

Follow these steps:

- Create a basic HTML file.
- 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 five 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 five 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 expense 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 disposable income they would like to put into savings.
- Finally, create an alert to display the total 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.

