JavaScript Additional Notes

JavaScript Output

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using innerHTML.
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

Using innerHTML

To access an HTML element, JavaScript can use the document.getElementById(id) method.
The id attribute defines the HTML element. The innerHTML property defines the HTML content:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
</body>
</html>
```

Changing the innerHTML property of an HTML element is a common way to display data in HTML.

Using document.write()

For testing purposes, it is convenient to use document.write():Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
document.write(5 + 6);
</script>
</body>
</html>
```

Using document.write() after an HTML document is loaded, will **delete all existing HTML**:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<button type="button" onclick="document.write(5 + 6)">Try it</button>
</body>
</html>
```

The document.write() method should only be used for testing.

Using window.alert()

You can use an alert box to display data:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
window.alert(5 + 6);
</script>
</body>
</html>
```

You can skip the window keyword.

In JavaScript, the window object is the global scope object. This means that variables, properties, and methods by default belong to the window object. This also means that specifying the window keyword is optional:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
alert(5 + 6);
</script>
</body>
</html>
```

Using console.log()

For debugging purposes, you can call the console.log() method in the browser to display data.

You will learn more about debugging in a later chapter.

Example

```
<!DOCTYPE html>
<html>
<body>
<script>
```

```
console.log(5 + 6);
</script>
</body>
</html>
```

JavaScript Print

JavaScript Colleges not have any print object or print methods.

You cannot access output devices from JavaScript.

The only exception is that you can call the window.print() method in the browser to print the content of the current window.

Example

```
<!DOCTYPE html>
<html>
<body>
<button onclick="window.print()">Print this page</button>
</body>
</html>
```

JavaScript Input

The prompt() method displays a dialog box that prompts the user for input.

The prompt() method returns the input value if the user clicks "OK", otherwise it returns null.

Note

A prompt box is used if you want the user to input a value.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed.

Do not overuse this method. It prevents the user from accessing other parts of the page until the box is closed.

```
<html>
<body>
<h1>The Window Object</h1>
<h2>The prompt() Method</h2>
<p>Click the button to demonstrate the prompt box.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  let person = prompt("Please enter your name", "Harry Potter");
  if (person != null) {
    document.getElementById("demo").innerHTML =
    "Hello " + person + "! How are you today?";
  }
}
</script>
</body>
</html>
```

```
var name = prompt("What is your name?");
var num = prompt("What is your favorite number? ");
```

```
// Uses user input to print out information
println("Hello " + name + "!");
println(num + "?! That's my favorite number too!");

// Prints out the variable type
println("Name is a " + typeof name);
println("Num is a " + typeof num);
```

Objects are Variables

JavaScript variables can contain single values:

Example

```
let person = "Sangola College";
```

JavaScript variables can also contain many values.

Objects are variables too. But objects can contain many values.

Object values are written as **name : value** pairs (name and value separated by a colon).

```
<body>

<h2>JavaScript Objects</h2>
<p>Creating an object:</p>
<p id="demo"></p>
<script>

let person = {
  firstName : "Sangola",
  lastName  : "College",
  age     : 50,
  eyeColor  : "blue"
};
document.getElementById("demo").innerHTML = person.firstName + " " + person.lastName;
</script>
</body>
</html>
```

A JavaScript object is a collection of **named values**

It is a common practice to declare objects with the const keyword.

Example

```
<html>
<body>

<h2>JavaScript Objects</h2>
<p>Creating an object:</p>
<p id="demo"></p>
<script>
const person = {
  firstName : "Sangola",
  lastName  : "College",
  age     : 50,
```

```
  eyeColor  : "blue"
};

document.getElementById("demo").innerHTML = person.firstName + " " + person.lastName;
</script>
</body>
</html>
```

Objects written as name value pairs are similar to:

- Associative arrays in PHP
- Dictionaries in Python
- Hash tables in C
- Hash maps in Java
- Hashes in Ruby and Perl

Object Methods

Methods are **actions** that can be performed on objects.

Object properties can be both primitive values, other objects, and functions.

An **object method** is an object property containing a **function definition**.

Property	Value
firstName	Sangola
lastName	College
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

JavaScript objects are containers for named values, called properties and methods.

Creating a JavaScript Object

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

- Create a single object, using an object literal.
- Create a single object, with the keyword new.
- Define an object constructor, and then create objects of the constructed type.
- Create an object using Object.create().

- **Using an Object Literal**

This is the easiest way to create a JavaScript Object.

Using an object literal, you both define and create an object in one statement.

An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

The following example creates a new JavaScript object with four properties:

- Example

```
const person = {firstName:"Sangola", lastName:"College", age:50, eyeColor:"blue"};
```

This example creates an empty JavaScript object, and then adds 4 properties:

- Example

```
const person = {};
person.firstName = "Sangola";
person.lastName = "College";
person.age = 50;
person.eyeColor = "blue";
```

- Using the JavaScript Keyword new

The following example create a new JavaScript object using new Object(), and then adds 4 properties:

- Example

```
const person = new Object();
person.firstName = "Sangola";
person.lastName = "College";
person.age = 50;
person.eyeColor = "blue";
```

```
<html>
<body>
<h2>JavaScript Objects</h2>
<p>Creating a JavaScript Object:</p>
<p id="demo"></p>
<script>
const person = new Object();
person.firstName = "Sangola";
person.lastName = "College";
person.age = 50;
person.eyeColor = "blue";
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
</body>
</html>
```

JavaScript Objects are Mutable

Objects are mutable: They are addressed by reference, not by value.

If person is an object, the following statement will not create a copy of person:

```
const x = person;  // Will not create a copy of person.
```

The object x is **not a copy** of person. It **is** person. Both x and person are the same object.

Any changes to x will also change person, because x and person are the same object.

```
<html>
<body>
```

```
<h2>JavaScript Objects</h2>
<p>JavaScript objects are mutable.</p>
<p>Any changes to a copy of an object will also change the original object:</p>
<p id="demo"></p>
<script>
const person = {
  firstName: "Sangola",
  lastName: "College",
  age:50,
  eyeColor: "blue"
};
const x = person;
x.age = 10;
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
</body>
</html>
```

JavaScript Object Properties

Properties are the most important part of any JavaScript object.

JavaScript Properties

Properties are the values associated with a JavaScript object.
A JavaScript object is a collection of unordered properties.
Properties can usually be changed, added, and deleted, but some are read only.

Accessing JavaScript Properties

The syntax for accessing the property of an object is:

```
objectName.property      // person.age
or
objectName["property"]   // person["age"]
or
objectName[expression]   // x = "age"; person[x]
```

The expression must evaluate to a property name.

```
<html>
<body>
<h2>JavaScript Object Properties</h2>
<p>Looping object property values:</p>
<p id="demo"></p>
<script>
const person = {
  fname:"Sangola",
  lname:"College",
  age:25
};
```

```
let txt = "";
for (let x in person) {
  txt += person[x] + " ";
}
document.getElementById("demo").innerHTML = txt;
</script>
</body>
</html>
```

. Adding New Properties

You can add new properties to an existing object by simply giving it a value.
Assume that the person object already exists - you can then give it new properties:

```
<html>
<body>
<h2>JavaScript Object Properties</h2>
<p>Add a new property to an existing object:</p>
<p id="demo"></p>
<script>
const person = {
  firstname: "Sangola",
  lastname: "College",
  age: 50,
  eyecolor: "blue"
};
person.nationality = "English";
document.getElementById("demo").innerHTML =
person.firstname + " is " + person.nationality + ".";
</script>
</body>
</html>
```

Deleting Properties

The delete keyword deletes a property from an object:

```
const person = {
  firstName: "Sangola",
  lastName: "College",
  age: 50,
  eyeColor: "blue"
};
delete person.age;
```

- The delete keyword deletes both the value of the property and the property itself.
- After deletion, the property cannot be used before it is added back again.
- The delete operator is designed to be used on object properties. It has no effect on variables or functions.
- The delete operator should not be used on predefined JavaScript object properties. It can crash your application.

Nested Objects

Values in an object can be another object:

Example

```
<html>
<body>
<h2>JavaScript Objects</h2>
<p>Access nested objects:</p>
<p id="demo"></p>
<script>
const myObj = {
  name: "Sangola",
  age: 30,
  cars: {
  car1: "Ford",
  car2: "BMW",
  car3: "Fiat"
  }
}
document.getElementById("demo").innerHTML = myObj.cars.car2;
</script>

</body>
</html>
```

Nested Arrays and Objects

Values in objects can be arrays, and values in arrays can be objects:

Example

```
const myObj = {
  name: "Sangola",
  age: 30,
  cars: [
    {name:"Ford", models:["Fiesta", "Focus", "Mustang"]},
    {name:"BMW", models:["320", "X3", "X5"]},
    {name:"Fiat", models:["500", "Panda"]}
  ]
}
```

```
<html>
<body>
<h1>JavaScript Arrays</h1>
<h2>Nested JavaScript Objects and Arrays.</h2>
<p id="demo"></p>
<script>
let x = "";
const myObj = {
  name: "Sangola",
```

```
  age: 30,
  cars: [
    {name:"Ford", models:["Fiesta", "Focus", "Mustang"]},
    {name:"BMW", models:["320", "X3", "X5"]},
    {name:"Fiat", models:["500", "Panda"]}
  ]
}
for (let i in myObj.cars) {
  x += "<h2>" + myObj.cars[i].name + "</h2>";
  for (let j in myObj.cars[i].models) {
    x += myObj.cars[i].models[j] + "<br>";
  }
}
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

Property Attributes

All properties have a name. In addition they also have a value.
The value is one of the property's attributes.
Other attributes are: enumerable, configurable, and writable.
These attributes define how the property can be accessed (is it readable?, is it writable?)
In JavaScript, all attributes can be read, but only the value attribute can be changed (and only if the property is writable).
(ECMAScript 5 has methods for both getting and setting all property attributes)

Prototype Properties

JavaScript objects inherit the properties of their prototype.
The delete keyword Colleges not delete inherited properties, but if you delete a prototype property, it will affect all objects inherited from the prototype.

JavaScript **Object Methods**

```
<html>
<body>

<h1>The JavaScript <i>this</i> Keyword</h1>
<p>In this example, <b>this</b> refers to the <b>person</b> object.</p>
<p>Because <b>fullName</b> is a method of the person object.</p>
<p id="demo"></p>
<script>
// Create an object:
const person = {
  firstName: "Sangola",
  lastName: "College",
  id: 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
```

```
};
// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName();
</script>
</body>
</html>
```

What is this?

In JavaScript, the this keyword refers to an **object**.
Which object depends on how this is being invoked (used or called).
The this keyword refers to different objects depending on how it is used:

In an object method, this refers to the **object**.
Alone, this refers to the **global object**.
In a function, this refers to the **global object**.
In a function, in strict mode, this is undefined.
In an event, this refers to the **element** that received the event.
Methods like call(), apply(), and bind() can refer this to **any object**.

Note

this is not a variable. It is a keyword. You cannot change the value of **this**.

Accessing Object Methods

You access an object method with the following syntax:
objectName.methodName()
You will typically describe fullName() as a method of the person object, and fullName as a property.
The fullName property will execute (as a function) when it is invoked with ().
This example accesses the fullName() **method** of a person object:
If you access the fullName **property**, without (), it will return the **function definition**:

```
<html>
<body>

<h2>JavaScript Objects</h2>
<p>Creating and using an object method.</p>
<p>A method is actually a function definition stored as a property value.</p>
<p id="demo"></p>
<script>
const person = {
  firstName: "Sangola",
  lastName: "College",
  id: 5566,
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};
document.getElementById("demo").innerHTML = person.fullName();
</script>
</body>
```

```
</html>
```

Adding a Method to an Object

Adding a new method to an object is easy:

```
<html>
<body>
<h2>JavaScript Objects</h2>
<p id="demo"></p>
<script>
const person = {
  firstName: "Sangola",
  lastName: "College",
  id: 5566,
};
person.name = function() {
  return this.firstName + " " + this.lastName;
};
document.getElementById("demo").innerHTML =
"My father is " + person.name();
</script>
</body>
</html>
```

Using Built-In Methods

This example uses the toUpperCase() method of the String object, to convert a text to uppercase:

```
<html>
<body>
<h2>JavaScript Objects</h2>
<p id="demo"></p>
<script>
const person = {
  firstName: "Sangola",
  lastName: "College",
  id: 5566,
};
person.name = function() {
  return (this.firstName + " " + this.lastName).toUpperCase();
};
document.getElementById("demo").innerHTML =
"My father is " + person.name();
</script>
</body>
</html>
```

How to Display JavaScript Objects?

Displaying a JavaScript object will output **[object Object]**.

Some common solutions to display JavaScript objects are:

- Displaying the Object Properties by name
- Displaying the Object Properties in a Loop
- Displaying the Object using Object.values()

- Displaying the Object using JSON.stringify()

Displaying Object Properties

The properties of an object can be displayed as a string:

```
<html>
<body>

<h2>JavaScript Objects</h2>
<p>Display object properties:</p>

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

<script>
const person = {
  name: "Sangola",
  age: 30,
  city: "New York"
};

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

</body>
</html>
```

Using Object.values()

Any JavaScript object can be converted to an array using Object.values():

```
<html>
<body>
<h2>JavaScript Objects</h2>
<p>Object.values() converts an object to an array.</p>
<p id="demo"></p>
<script>
const person = {
  name: "Sangola",
  age: 30,
  city: "New York"
};
document.getElementById("demo").innerHTML = Object.values(person);
</script>
</body>
</html>
```

Using JSON.stringify()

Any JavaScript object can be stringified (converted to a string) with the JavaScript function JSON.stringify():

```
<html>
<body>
<h2>JavaScript Objects</h2>
<p>Display properties in JSON format:</p>
```

```
<p id="demo"></p>
<script>
const person = {
  name: "Sangola",
  age: 30,
  city: "New York"
};
document.getElementById("demo").innerHTML = JSON.stringify(person);
</script>
</body>
</html>
```

JavaScript Accessors (Getters and Setters)

ECMAScript 5 (ES5 2009) introduced Getter and Setters.
Getters and setters allow you to define Object Accessors (Computed Properties).

JavaScript Getter (The get Keyword)

This example uses a lang property to get the value of the language property.

```
<html>
<body>

<h2>JavaScript Getters and Setters</h2>

<p>Getters and setters allow you to get and set object properties via methods.</p>

<p>This example uses a lang property to get the value of the language property:</p>

<p id="demo"></p>
<script>
// Create an object:
const person = {
  firstName: "Sangola",
  lastName: "College",
  language: "en",
  get lang() {
    return this.language;
  }
};
// Display data from the object using a getter:
document.getElementById("demo").innerHTML = person.lang;
</script>
</body>
</html>
```

JavaScript Setter (The set Keyword)

This example uses a lang property to set the value of the language property.

```
<html>
<body>
<h2>JavaScript Getters and Setters</h2>
```

```
<p>Getters and setters allow you to get and set properties via methods.</p>

<p>This example uses a lang property to set the value of the language property.</p>

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

<script>
// Create an object:
const person = {
  firstName: "Sangola",
  lastName: "College",
  language: "NO",
  set lang(value) {
    this.language = value;
  }
};

// Set a property using set:
person.lang = "en";

// Display data from the object:
document.getElementById("demo").innerHTML = person.language;
</script>

</body>
</html>
```

Why Using Getters and Setters?

- It gives simpler syntax
- It allows equal syntax for properties and methods
- It can secure better data quality
- It is useful for doing things behind-the-scenes

JavaScript Object Constructors

```
<html>
<body>

<h2>JavaScript Object Constructors</h2>

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

<script>
// Constructor function for Person objects
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}

// Create a Person object
```

```
const myFather = new Person("Sangola", "College", 50, "blue");

// Display age
document.getElementById("demo").innerHTML =
"My father is " + myFather.age + ".";
</script>

</body>
</html>
```