

Internet Technologies

JavaScript

Mahesh Kumar

[maheshkumar@andc.du.ac.in]

Course Web Page

[www.mkbhandari.com/mkwiki]

Outline

- 1 Basic JavaScript Instructions
- 2 Functions, Methods & Objects
- 3 Decisions and Loops [Self Study]
- 4 Document Object Model
- 5 Events

Statements


- A **script** is a series of instructions that a computer can follow one-by-one.
- Each individual instruction or step is known as a **statement**.
- Statements should end with a semicolon.

Statements

- A **script** is a series of instructions that a computer can follow one-by-one.
- Each individual instruction or step is known as a **statement**.
- Statements should end with a semicolon.

```
var today= new Date();  
var hourNow = today.getHours() ;  
var greeting;
```

```
if (hourNow > 18) {  
    greeting= 'Good evening';  
} else if (hourNow > 12) {  
    greeting= 'Good afternoon';  
} else if (hourNow > 0) {  
    greeting 'Good morning';  
} else {  
    greeting 'Welcome';  
}  
document.write(greeting);
```

 *Identify statements, code blocks, conditionals in the above lines of code?*

Statements

- JavaScript is **case sensitive**
 - ***hourNow** means something different to **HourNow** or **HOURLNOW**.*
- Statements are instructions and each one **starts on a new line**
 - *A statement is an **individual instruction** that the computer should follow.*
 - *Each one should start on a new line and **end with a semicolon**.*
 - *makes your **code easier to read and follow***
 - *The semicolon also tells the JavaScript interpreter when a **step is over** and **move to the next step**.*
- Statements can be **organized into code blocks**
 - *Group together many **related statements**, helps the programmers organize their code and makes it more readable.*

Comments

- You should write comments to explain what your code does.
 - *help make your code easier to read and understand.*
 - *help you and others who read your code.*

Comments

- You should write comments to explain what your code does.

- *help make your code easier to read and understand.*
- *help you and others who read your code.*
- **multi-line** and **single-line** comments are available in JavaScript.

```
/* This script displays a greeting to the user based upon the  
current time */
```

```
var today= new Date();           // Create a new date object  
var hourNow = today.getHours(); // Find the current hour  
var greeting;
```

```
// Display the appropriate greeting based on the current time  
if (hourNow > 18) {  
    greeting= 'Good evening';  
} else if (hourNow > 12) {  
    greeting= 'Good afternoon';  
} else if (hourNow > 0) {  
    greeting 'Good morning';  
} else {  
    greeting 'Welcome';  
}  
document.write(greeting);
```

What is a variable?

- Store values that can be used in other parts of the program and the data stored in a variable can change(or vary) each time a script runs.
 - To **create a variable** in JavaScript, use the **var** keyword.
 - **var quantity;** // declares the variable, value is **undefined**
 - **quantity = 3;** // assigns a value to the variable
- 1 *var is a **keyword**, is used to create variables*
- 2 *quantity is an **identifier***
- 3 *If a variable name is more than one word, it is usually written in **camelCase***
- 4 *A variable's name should describe the kind of data it holds.*
- 5 *Where a variable is described can have an effect upon whether the rest of the script can use it, called as the **scope of a variable***

Data Types

- JavaScript distinguishes between **numbers**, **strings**, and true or false values known as **Booleans**.

1 *Numeric Data Types*

- *numeric data type handles numbers.*
- For example: **2022**, **-25000**, **0.75**

2 *String Data Types*

- *strings data type consists of letters and other characters.*
- For example: **"ANDC College"**, **'Delhi'**
- *Strings can be used when working with any kind of text.*

Data Types

- JavaScript distinguishes between **numbers**, **strings**, and true or false values known as **Booleans**.

3 *Boolean Data Types*

→ *Boolean data types can have one of two values: **true** or **false**.*

- In addition to these three data types, JavaScript also has others (**arrays**, **objects**, **undefined**, and **null**)
- Unlike some other programming languages, when declaring a variable in JavaScript, you do not need to specify what type of data it will hold.

Using a variable to store a Number

JAVASCRIPT

c02/js/numeric-variable.js

```
var price;  
var quantity;  
var total;  
  
price = 5;  
quantity = 14;  
total = price * quantity;  
  
var el = document.getElementById('cost');  
el.textContent = '$' + total;
```

Using a variable to store a String

c02/js/string-variable.js

JAVASCRIPT

```
var username;  
var message;  
username = 'Molly';  
message = 'See our upcoming range';  
  
var elName = document.getElementById('name');  
elName.textContent = username;  
var elNote = document.getElementById('note');  
elNote.textContent = message;
```

Using a variable to store a Boolean

c02/js/boolean-variable.js

JAVASCRIPT

```
var inStock;  
var shipping;  
inStock = true;  
shipping = false;  
  
var elStock = document.getElementById('stock');  
elStock.className = inStock;  
  
var elShip = document.getElementById('shipping');  
elShip.className = shipping;
```

c02/boolean-variable.html

HTML

```
<h1>Elderflower</h1>  
<div id="content">  
  <div class="message">Available:  
    <span id="stock"></span></div>  
  <div class="message">Shipping:  
    <span id="shipping"></span></div>  
</div>  
<script src="js/boolean-variable.js"></script>
```

Shorthand for creating variables

JAVASCRIPT

c02/js/shorthand-variable.js

① `var price = 5;
var quantity = 14;
var total = price * quantity;`

② `var price, quantity, total;
price = 5;
quantity = 14;
total = price * quantity;`

③ `var price = 5, quantity = 14;
var total = price * quantity;`

④ `// Write total into the element with id of cost
var el = document.getElementById('cost');
el.textContent = '$' + total;`

Shorthand for creating variables

JAVASCRIPT

c02/js/shorthand-variable.js

① `var price = 5;
var quantity = 14;
var total = price * quantity;`

② `var price, quantity, total;
price = 5;
quantity = 14;
total = price * quantity;`

③ `var price = 5, quantity = 14;
var total = price * quantity;`

④ `// Write total into the element with id of cost
var el = document.getElementById('cost');
el.textContent = '$' + total;`

Rules for Naming Variables

- Here are **six rules** you must always follow when giving a variable a name:
 - 1 The name must begin with a **letter**, **dollar sign (\$)**, or an **underscore (_)**. It must not start with a number.
 - 2 The name can contain **letters**, **numbers**, **dollar sign (\$)**, or an **underscore (_)**. Note that you must not use a dash (-) or a period (.) in a variable name.
 - 3 You cannot use **keywords** or **reserved words**. Keywords are special words that tell the interpreter to do something. For example, **var** is a keyword used to declare a variable. Reserved words are ones that may be used in a future version of JavaScript.
 - 4 All variables are **case sensitive**, so **score** and **Score** would be different variable names, but it is bad practice to create two variables that have the same name using different cases.

Rules for Naming Variables

- Here are **six rules** you must always follow when giving a variable a name:
 - 5 ***Use a name that describes the kind of information that the variable stores.*** For example, `firstName` might be used to store a person's first name, `lastName` for their last name, and `age` for their age.
 - 6 ***If your variable name is made up of more than one word, use **camel case**. For example, `firstName` rather than `firstname`. You can also use an **underscore** between each word (you cannot use a dash).***

Arrays

- An array is a special type of variable, stores a **list of values**.
 - *consider using an array whenever you are working with a **list or a set of values that are related to each other**.*
 - *Arrays are especially helpful when you **do not know how many items a list will contain** because, when you create the array, you do not need to specify how many values it will hold.*
- Arrays can be created in two ways:
 - 1 **array literal**
 - 2 **array constructor**

Creating an Array

1

JAVASCRIPT

c02/js/array-literal.js

```
var colors;  
colors = ['white', 'black', 'custom'];  
  
var el = document.getElementById('colors');  
el.textContent = colors[0];
```

2

JAVASCRIPT

c02/js/array-constructor.js

```
var colors = new Array('white',  
                        'black',  
                        'custom');  
  
var el = document.getElementById('colors');  
el.innerHTML = colors.item(0);
```

Values in Arrays

- Values in an array are accessed as if they are in a numbered list.
 - *The numbering of this list starts at **zero** (not one).*

1 Numbering items in an array

- *Each item in an array is automatically given a number called an **index**.*
- *can be used to access specific items in the array.*

```
var colors;  
colors = ['white',  
         'black',  
         'custom'];
```

INDEX	VALUE
0	'white'
1	'black'
2	'custom'

Values in Arrays

2 Accessing items in an array

- To retrieve a particular item from the list, the array name is specified along with the **index number in square brackets**.

```
var itemThree;  
itemThree = colors[2];
```

3 Number of items in an Array

- Each array has a property called **length**, which holds the number of items in the array

```
var numColors;  
numColors = colors.length;
```

Accessing and Changing values in an Arrays

JAVASCRIPT

c02/js/update-array.js

```
// Create the array
var colors = ['white',
              'black',
              'custom'];

// Update the third item in the array
colors[2] = 'beige';

// Get the element with an id of colors
var el = document.getElementById('colors');

// Replace with third item from the array
el.textContent = colors[2];
```

Expressions

- An expression evaluates into (results in) a single value. There are two types of expressions.

1 Expressions that just Assign a value to a variable

- *In order for a variable to be useful, it needs to be given a value.*
- *When you first declare a variable using the var keyword, it is given a special value of **undefined**, and it changes when you assign a value to it.*

```
var color = 'beige';
```

2 Expressions that use two or more values to return a single value

- *You can perform operations on any number of individual values to determine a single value. For example:*

```
var area = 3 * 2;
```

Operators

- **Expressions rely on things called operators;** allow programmers to create a single value from one or more values.

- Arithmetic Operators [+ , - , * , / , % , ++ , --]
- Logical Operators [&& , || , !]
- Comparison Operators [== , === , != , !== , < , <= , > , >=]
- String Operators [+]
- Assignment Operators [= , += , -= , *= , /= , %=]
- The Conditional Expression Ternary Operator [**condition ? value1 : value2**]
- The delete Operator: Delete a **property of an object** or an **array element**.
- The new Operator: Create an **instance of an object type**.
- The void Operator: returns a **URL** with no value.

Functions

- Functions let you group a series of statements together to perform a specific task. If different parts of a script repeat the same task, you can reuse the function
 - *The statements in a function are not always executed when a page loads, so functions also offer a way to store the steps needed to achieve a task.*
 - *When you ask it to perform its task, it is known as **calling** the function.*
 - *The steps that the function needs to perform in order to perform its task are packaged up in a **code block**.*
 - *Some functions need to be provided with information in order to achieve a given task are known as **parameters**.*
 - *When you write a function and you expect it to provide you with an answer, the response is known as a **return value**.*

Functions

HTML

c03/basic-function.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Basic Function</title>
    <link rel="stylesheet" href="css/c03.css" />
  </head>
  <body>
    <h1>TravelWorthy</h1>
    <div id="message">Welcome to our site!</div>
    <script src="js/basic-function.js"></script>
  </body>
</html>
```

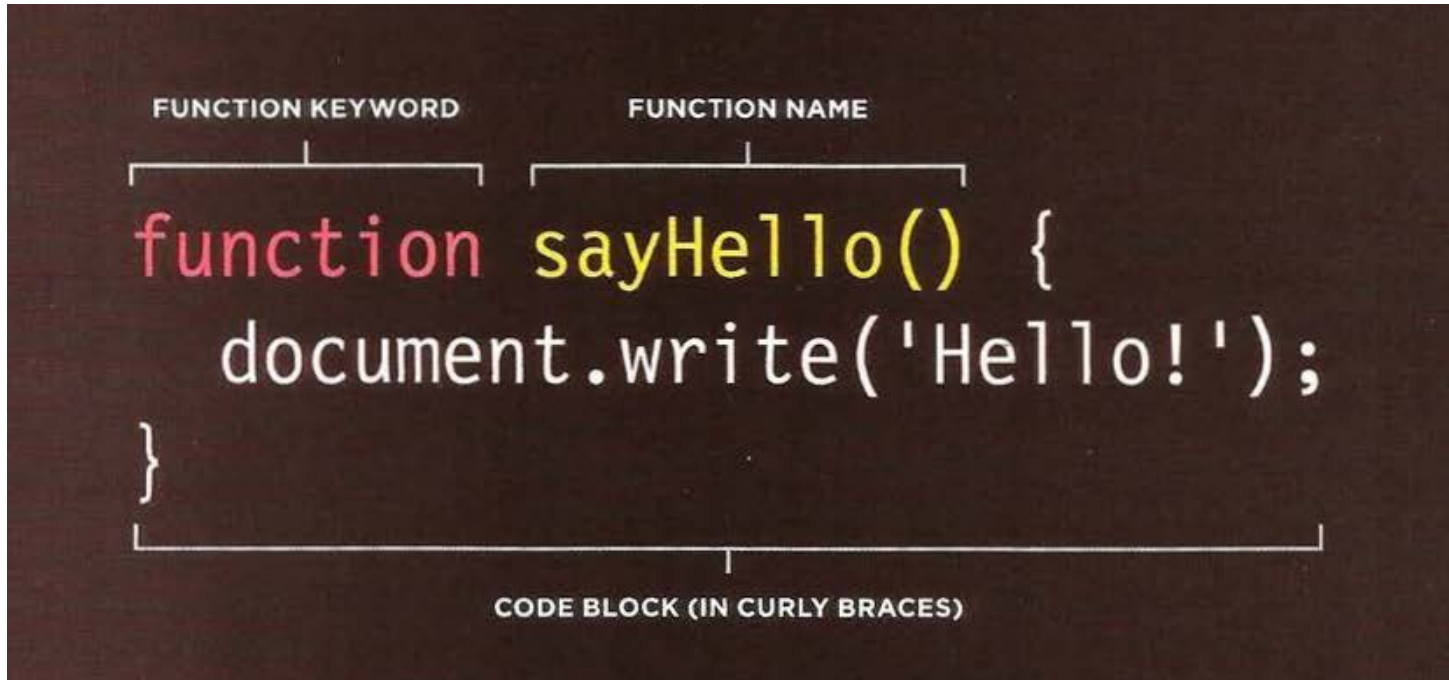
JAVASCRIPT

c03/js/basic-function.js

```
var msg = 'Sign up to receive our newsletter for 10% off!';
function updateMessage() {
  var el = document.getElementById('message');
  el.textContent = msg;
}
updateMessage();
```

Declaring a Functions

- To create a function, you give it a name and then write the statements needed to achieve its task inside the curly braces. This is known as a **function declaration**.

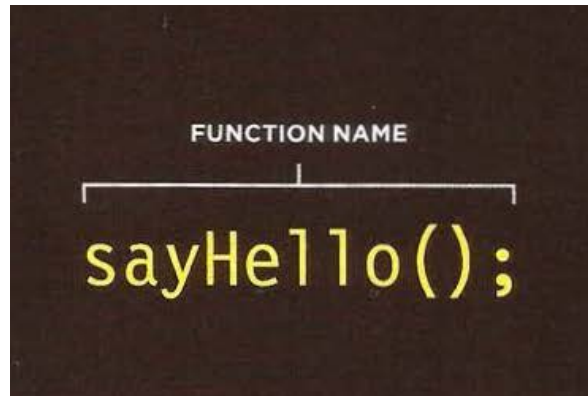


```
function sayHello() {  
    document.write('Hello!');  
}
```

The diagram illustrates the components of a function declaration. The word `function` is labeled as the **FUNCTION KEYWORD**. The text `sayHello()` is labeled as the **FUNCTION NAME**. The entire block of code, including the curly braces and the statement `document.write('Hello!');`, is labeled as the **CODE BLOCK (IN CURLY BRACES)**.

Calling a Functions

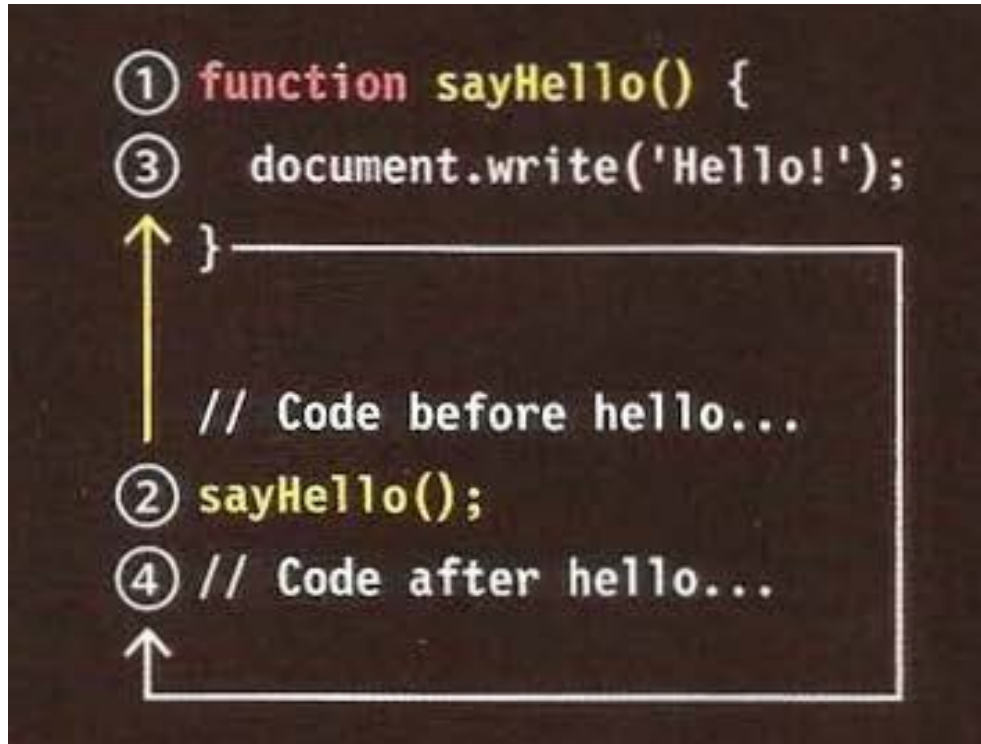
- Having declared the function, you can then execute all of the statements between its curlybraces with just one line of code. This is known as **calling the function**.



A diagram on a dark background showing the text "FUNCTION NAME" in white. Below it, a horizontal line with vertical end caps is positioned above the text "sayHello();" in yellow. A vertical line connects the center of the horizontal line to the "FUNCTION NAME" text above, illustrating that "sayHello();" is the call to the function named "FUNCTION NAME".

- You can call the same function as many times as you want within the same JavaScript file.

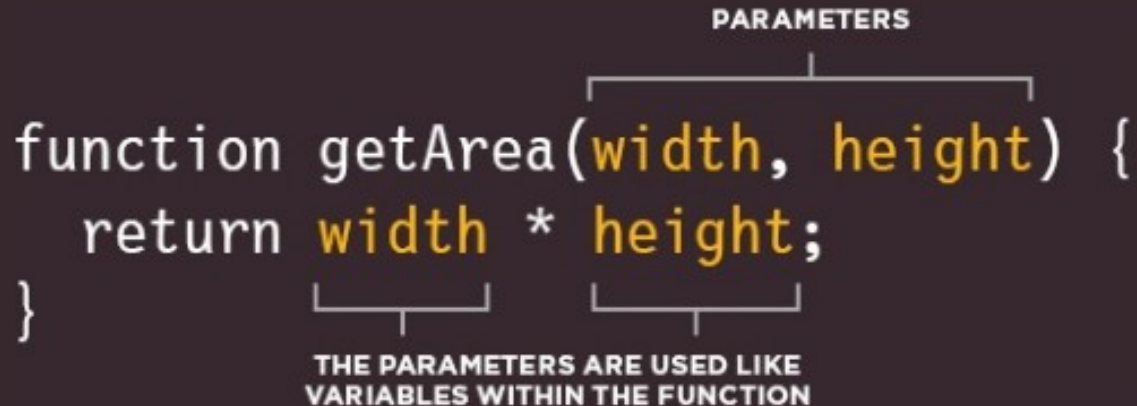
Calling a Functions



- 1 The function can store the instructions for a specific task.
- 2 When you need the script to perform that task, you call the function.
- 3 The function executes the code in that code block.
- 4 When it has finished, the code continues to run from the point where it was initially called.

Declaring functions that need information

- Sometimes a function needs specific information to perform its task. In such cases, when you declare the function you give it **parameters**. Inside the function, the parameters act like variables.



```
function getArea(width, height) {  
    return width * height;  
}
```

The diagram illustrates the role of parameters in a function. A bracket above the function signature `width, height` is labeled "PARAMETERS". Inside the function body, the variables `width` and `height` are used in the expression `width * height`. Brackets below these variables point to a text box that states: "THE PARAMETERS ARE USED LIKE VARIABLES WITHIN THE FUNCTION".

Calling functions that need information

- When you call a function that has parameters, you specify the values it should use in the parentheses that follow its name. The values are called **arguments**, and they can be provided as values or as variables.

1 Arguments as Values

```
getArea(3, 5);
```

2 Arguments as Variables

```
wallWidth = 3;  
wallHeight = 5;  
getArea(wallWidth, wallHeight);
```

Getting a single value out of a function

- Some functions return information to the code that called them. For example, when they perform a calculation, they return the result.

```
function calculateArea(width, height) {  
    var area = width * height;  
    return area;  
}  
  
var wallOne = calculateArea(3, 5);  
var wallTwo = calculateArea(8, 5);
```


Getting multiple values out of a function

- Functions can return more than one value using an array. For example, this function calculates the area and volume of a box.

```
function getSize(width, height, depth) {  
    var area = width * height;  
    var volume = width * height * depth;  
    var sizes = [area, volume];  
    return sizes;  
}  
  
var areaOne = getSize(3, 2, 3)[0];  
var volumeOne = getSize(3, 2, 3)[1];
```

Anonymous Functions & Function Expressions

- Expressions produce a value. They can be used where values are expected. If a function is placed where a browser expects to see an expression, (e.g., as an argument to a function), then it gets treated as an expression.
- A **function declaration** creates a function that you can call later in your code.
- In order to call the function later in your code, **you must give it a name**, so these are known as **named functions**.

```
function area(width, height) {  
    return width * height;  
};  
  
var size = area(3, 4);
```

Anonymous Functions & Function Expressions

- Interpreter always looks for variables and function declarations before going through each section of a script, line-by-line.
 - *a function created with a function declaration can be called before it has even been declared.*
- If you put a function where the interpreter would expect to see an expression, then it is treated as an expression, known as a **function expression**
- A function with no name is called an **anonymous function**.

```
var area = function(width, height) {  
    return width * height;  
};  
  
var size = area(3, 4);
```

Immediately Invoked Function Expressions (IIFE)

- Pronounced "**iffy**" these functions are not given a name. Instead, they are **executed once** as the interpreter comes across them.
- The final parentheses (**shown in green**) tell the interpreter to call the function immediately.
- The grouping operators (**shown in red**) are parentheses there to ensure the interpreter treats this as an expression.
- IIFEs are commonly used as a **wrapper** around a set of code.
- It is also a very popular technique with **jQuery**.

```
var area = (function() {  
    var width = 3;  
    var height = 2;  
    return width * height;  
})();
```

Variable Scope

- The location where you declare a variable will affect where it can be used within your code. If you declare it within a function, it can only be used within that function. This is known as the variable's **scope**.

- 1 Local variables
- 2 Global variables

```
function getArea(width, height) {  
    var area = width * height;  
    return area;  
}  
  
var wallSize = getArea(3, 2);  
document.write(wallSize);
```



LOCAL (OR FUNCTION-LEVEL) SCOPE



GLOBAL SCOPE

Variable Scope

1 Local variables

- *When a variable is created inside a function using the var keyword, it can only be used in that function. It is said to have **local scope** or **function-level scope**.*
- *It cannot be accessed outside of the function in which it was declared.*
- *The interpreter creates local variables when the function is run, and removes them as soon as the function has finished its task.*

2 Global variables

- *If you create a variable outside of a function, then it can be used anywhere within the script. It is called a **global variable** and has **global scope**.*
- *stored in memory for as long as the web page is loaded into the web browser.*
- *take up more memory than local variables, and it also increases the **risk of naming conflicts***

What is an Object?

- Objects group together a set of variables and functions to create a **model** of a something you would recognize from the real world.
- In an object, variables and functions take on new names.
 - *If a variable is part of an object, it is called a **property** (represents features of the objects).*
 - **For example:** *name of a hotel or the number of rooms.* Each individual hotel might have a different name and a different number of rooms.
 - *If a function is part of an object, it is called a **method** (represents tasks associated with the objects).*
 - **For example:** *check how many rooms are available* by subtracting the number of booked rooms from the total number of rooms.

What is an Object?

- Like variables and named functions, properties and methods have a name and a value. In an object, that name is called a **key**.
- An object cannot have two keys with the same name
 - *keys are used to access their corresponding values.*
- The value of a property can be a string, number, boolean, array, or even another object.
- The value of a method is always a function.

What is an Object?

```
var hotel = {
```

● KEY
● VALUE

```
  name: 'Quay',  
  rooms: 40,  
  booked: 25,  
  gym: true,  
  roomTypes: ['twin', 'double', 'suite'],
```

PROPERTIES
These are variables

```
  checkAvailability: function() {  
    return this.rooms - this.booked;  
  }
```

METHOD
This is a function

```
};
```

[*Example: A hotel object*]

What is an Object?

- The hotel object contains the following **key/value** pairs:

PROPERTIES:	KEY	VALUE
	name	string
	rooms	number
	booked	number
	gym	Boolean
	roomTypes	array
METHODS:	checkAvailability	function

What is an Object?

- The hotel object contains the following **key/value** pairs:

PROPERTIES:	KEY	VALUE
	name	string
	rooms	number
	booked	number
	gym	Boolean
	roomTypes	array
METHODS:	checkAvailability	function

- 🟡 Q Identify the name/value pairs used in **HTML, CSS, JavaScript** ?

What is an Object?

Programmers use a lot of name/value pairs:

- HTML uses attribute names and values.
- CSS uses property names and values.

In JavaScript:

- Variables have a name and you can assign them a value of a string, number, or Boolean.
- Arrays have a name and a group of values. (Each item in an array is a name/value pair because it has an index number and a value.)
- Named functions have a name and value that is a set of statements to run if the function is called.
- Objects consist of a set of name/value pairs (but the names are referred to as keys).

Creating an Object: Literal Notation

- Most easiest and most popular way to create objects.

```
var hotel = {
```

```
  name: 'Quay',
```

```
  rooms: 40,
```

```
  booked: 25,
```

```
  checkAvailability: function() {  
    return this.rooms - this.booked;  
  }
```

```
};
```

● OBJECT
● KEY
● VALUE

PROPERTIES

METHOD

Accessing an Object and Dot Notation

- Accessing the properties or methods of an object using **dot notation**.



The diagram illustrates the syntax of dot notation with two example lines of code. The first line is `var hotelName = hotel.name;` and the second is `var roomsFree = hotel.checkAvailability();`. Above the first line, a bracket labeled "OBJECT" spans the word `hotel`, and another bracket labeled "PROPERTY/METHOD NAME" spans the `.name` part. In the second line, a bracket labeled "PROPERTY/METHOD NAME" spans the `.checkAvailability()` part. Below the second line, a vertical line labeled "MEMBER OPERATOR" points to the dot character between `hotel` and `checkAvailability()`.

```
var hotelName = hotel.name;
var roomsFree = hotel.checkAvailability();
```

- Accessing the properties using **square bracket syntax**.

```
var hotelName = hotel['name'];
```

Creating objects using Literal Notation

c3/js/object-literal.js

JAVASCRIPT

```
var hotel = {  
  name: 'Quay',  
  rooms: 40,  
  booked: 25,  
  checkAvailability: function() {  
    return this.rooms - this.booked;  
  }  
};  
  
var elName = document.getElementById('hotelName');  
elName.textContent = hotel.name;  
  
var elRooms = document.getElementById('rooms');  
elRooms.textContent = hotel.checkAvailability();
```


Creating more Object Literal

JAVASCRIPT

c03/js/object-literal2.js

```
var hotel = {  
  name: 'Park',  
  rooms: 120,  
  booked: 77,  
  checkAvailability: function() {  
    return this.rooms - this.booked;  
  }  
};  
  
var elName = document.getElementById('hotelName');  
elName.textContent = hotel.name;  
  
var elRooms = document.getElementById('rooms');  
elRooms.textContent = hotel.checkAvailability();
```


Creating an Object: Constructor Notation

- The **new** keyword and the **Object** constructor create a blank object.
 - *Then add properties and methods using dot notation.*

```
var hotel = new Object();
```

```
hotel.name = 'Quay';
```

```
hotel.rooms = 40;
```

```
hotel.booked = 25;
```

PROPERTIES

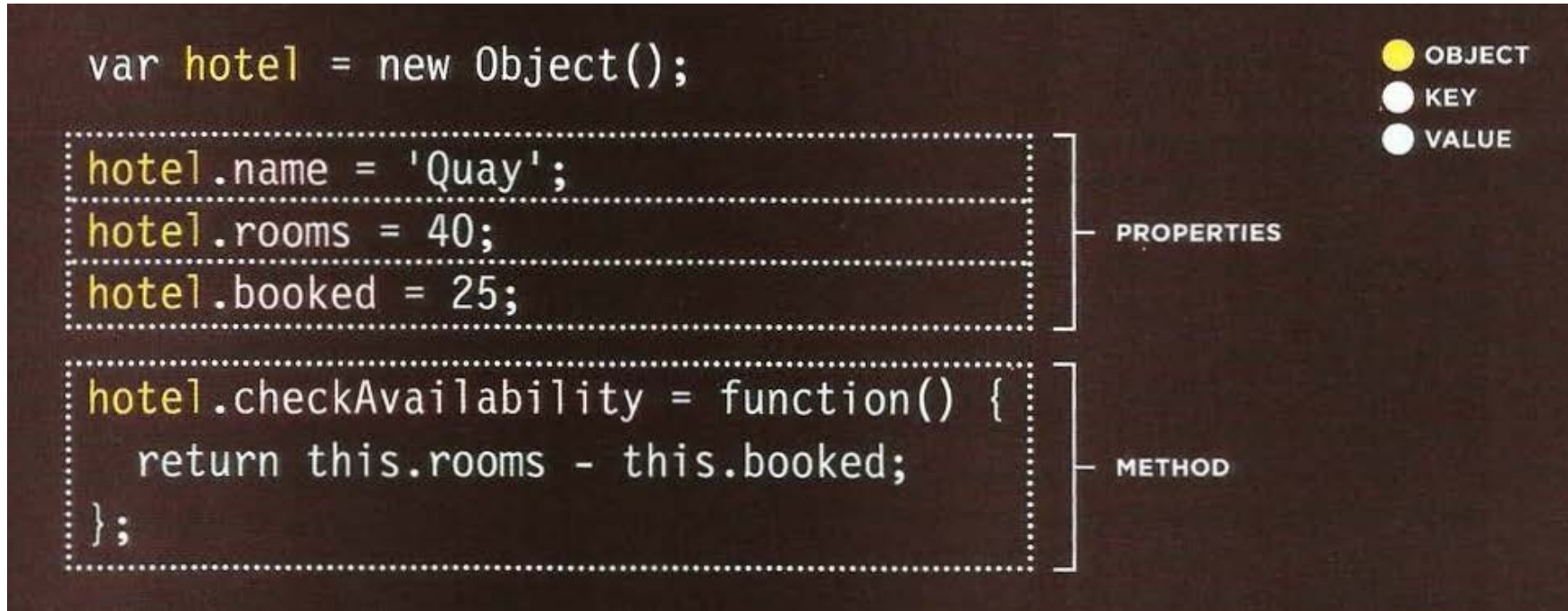
```
hotel.checkAvailability = function() {  
    return this.rooms - this.booked;  
};
```

METHOD

● OBJECT
○ KEY
○ VALUE

Creating an Object: Constructor Notation

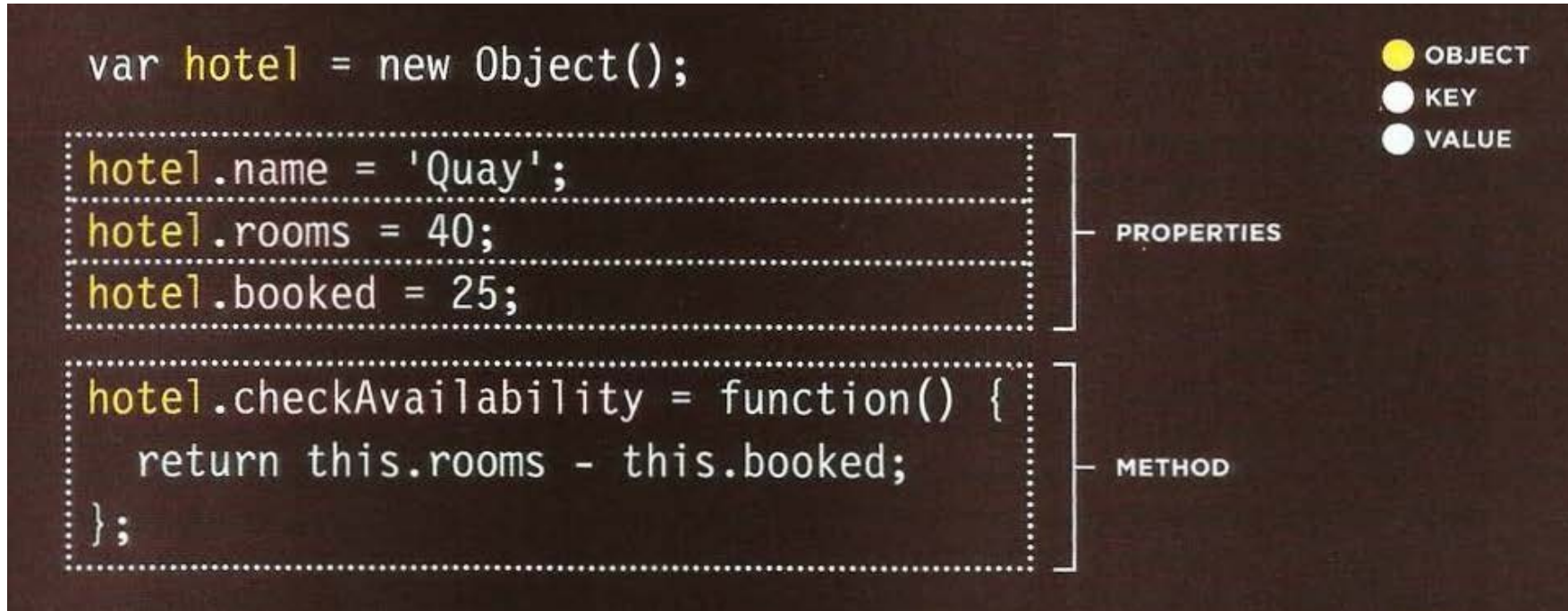
- The **new** keyword and the **Object** constructor create a blank object.
 - *Then add properties and methods using dot notation.*



Q How to create an empty object using Literal Notation

Creating an Object: Constructor Notation

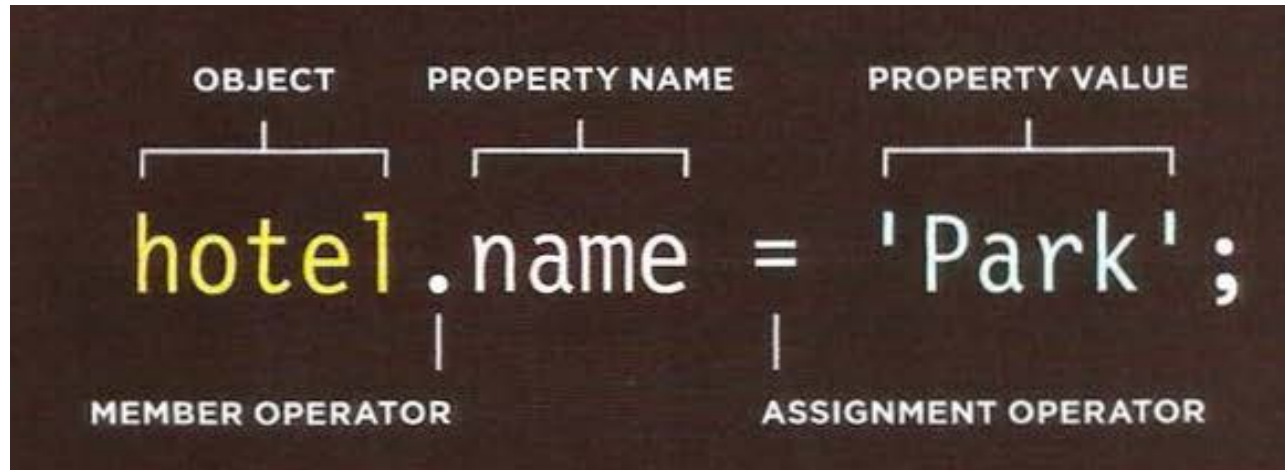
- The **new** keyword and the **Object** constructor create a blank object.
 - *Then add properties and methods using dot notation.*



Q How to create an empty object using Literal Notation **var hotel = {}**

Updating an Object

- To update the value of properties, use **dot notation or square brackets**
 - *Use the same technique as adding properties to the object, but give a new value.*
 - *If the object does not have the property you are trying to update, it will be added to the object.*



Updating an Object

→ *Properties of an object can be updated also using **square bracket syntax***

```
hotel['name'] = 'Park';
```

- To delete a property, use the **delete** keyword followed by the object name and property name.

```
delete hotel.name;
```

→ *To clear the value of a property*

```
hotel.name = '';
```


Creating many Objects: Constructor Notation

- Object constructors can use a function as a **template** for creating objects.
- First, create the template with the object's properties and methods.

```
function Hotel(name, rooms, booked) {  
  this.name = name;  
  this.rooms = rooms;  
  this.booked = booked;  
  
  this.checkAvailability = function() {  
    return this.rooms - this.booked;  
  };  
}
```

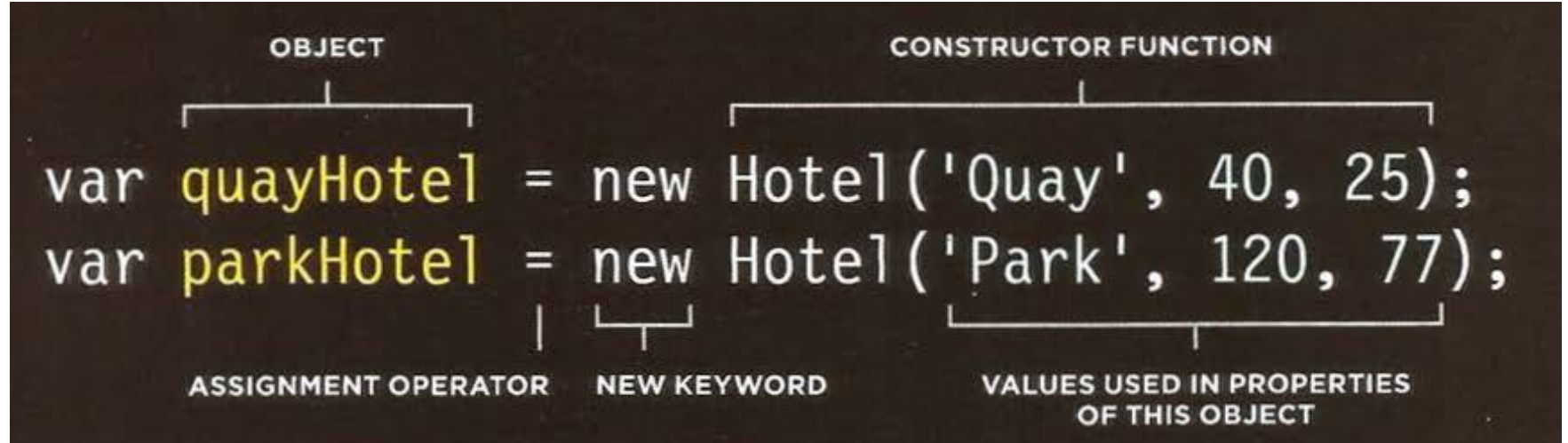
● KEY
● VALUE

PROPERTIES

METHOD

Creating many Objects: Constructor Notation

- Create **instances** of the object using the constructor function.
- The **new** keyword followed by a call to the function creates a new object
- The properties of each object are given as arguments to the function



Creating objects using Constructor Syntax

c3/js/object-constructor.js

JAVASCRIPT

```
var hotel = new Object();

hotel.name = 'Park';
hotel.rooms = 120;
hotel.booked = 77;
hotel.checkAvailability = function() {
    return this.rooms - this.booked;
};

var elName = document.getElementById('hotelName');
elName.textContent = hotel.name;

var elRooms = document.getElementById('rooms');
elRooms.textContent = hotel.checkAvailability();
```


Create & Access objects Constructor Notation

JAVASCRIPT

c03/js/multiple-objects.js

```
function Hotel(name, rooms, booked) {
  this.name = name;
  this.rooms = rooms;
  this.booked = booked;
  this.checkAvailability = function() {
    return this.rooms - this.booked;
  };
}

var quayHotel = new Hotel('Quay', 40, 25);
var parkHotel = new Hotel('Park', 120, 77);

var details1 = quayHotel.name + ' rooms: ';
  details1 += quayHotel.checkAvailability();
var elHotel1 = document.getElementById('hotel1');
elHotel1.textContent = details1;

var details2 = parkHotel.name + ' rooms: ';
  details2 += parkHotel.checkAvailability();
var elHotel2 = document.getElementById('hotel2');
elHotel2.textContent = details2;
```

Adding and Removing Properties

c3/js/adding-and-removing-properties.js

JAVASCRIPT

```
var hotel = {  
  name : 'Park',  
  rooms : 120,  
  booked : 77,  
};  
  
hotel.gym = true;  
hotel.pool = false;  
delete hotel.booked;  
  
var elName = document.getElementById('hotelName');  
elName.textContent = hotel.name;  
  
var elPool = document.getElementById('pool');  
elPool.className = 'Pool: ' + hotel.pool;  
  
var elGym = document.getElementById('gym');  
elGym.className = 'Gym: ' + hotel.gym;
```

Recap: Ways to Create Objects

- 1 Create the objects, then add properties & methods

LITERAL NOTATION

```
var hotel = {}  
  
hotel.name = 'Quay';  
hotel.rooms = 40;  
hotel.booked = 25;  
hotel.checkAvailability = function() {  
    return this.rooms - this.booked;  
};
```

OBJECT CONSTRUCTOR NOTATION

```
var hotel = new Object();  
  
hotel.name = 'Quay';  
hotel.rooms = 40;  
hotel.booked = 25;  
hotel.checkAvailability = function() {  
    return this.rooms - this.booked;  
};
```

Recap: Ways to Create Objects

2 Creating an object with properties & methods

LITERAL NOTATION

A colon separates the key/value pairs.

There is a comma between each key/value pair.

```
var hotel = {  
  name: 'Quay',  
  rooms: 40,  
  booked: 25,  
  checkAvailability: function() {  
    return this.rooms - this.booked;  
  }  
};
```

OBJECT CONSTRUCTOR NOTATION

The function can be used to create multiple objects.

The `this` keyword is used instead of the object name.

```
function Hotel(name, rooms, booked) {  
  this.name = name;  
  this.rooms = rooms;  
  this.booked = booked;  
  this.checkAvailability = function() {  
    return this.rooms - this.booked;  
  };  
}  
  
var quayHotel = new Hotel('Quay', 40, 25);  
var parkHotel = new Hotel('Park', 120, 77);
```

Arrays are Objects

- A special type of object, hold a related set of key/value pairs(lilke all objects), but the key for each value is its index number.

AN OBJECT

PROPERTY:

VALUE:

room1	:	420
room2	:	460
room3	:	230
room4	:	620

Here, hotel room costs are stored in an object. The example covers four rooms, and the cost for each room is a property of the object:

```
costs = {  
    room1: 420,  
    room2: 460,  
    room3: 230,  
    room4: 620  
};
```


Arrays are Objects

- A special type of object, hold a related set of key/value pairs(lilke all objects), but the key for each value is its index number.

AN ARRAY

INDEX NUMBER:

VALUE:

0	:	420
1	:	460
2	:	230
3	:	620

Here is the the same data in an array. Instead of property names, it has index numbers:

```
costs = [420, 460, 230, 620];
```

Arrays of Objects & Objects in Arrays

- Combine arrays and objects to create complex data structures:
 - *Arrays can store a series of objects(and remember their order)*
 - *Objects can also hold arrays(as value of their properties)*
 - *In an object the order in which the properties appear is not important.*
 - *In an array, the index number dictate the order of the properties.*

Arrays of Objects & Objects in Arrays

- Combine arrays and objects to create complex data structures:
 - *Arrays can store a series of objects(and remember their order)*
 - *Objects can also hold arrays(as value of their properties)*

ARRAYS IN AN OBJECT

The property of any object can hold an array.
On the left, each item on a hotel bill is stored separately in an array. To access the first charge for `room1` you would use:

```
costs.room1.items[0];
```

PROPERTY:

VALUE:

room1	:	items[420, 40, 10]
room2	:	items[460, 20, 20]
room3	:	items[230, 0, 0]
room4	:	items[620, 150, 60]

Arrays of Objects & Objects in Arrays

- Combine arrays and objects to create complex data structures:
 - *Arrays can store a series of objects(and remember their order)*
 - *Objects can also hold arrays(as value of their properties)*

OBJECTS IN AN ARRAY

The value of any element in an array can be an object (written using the object literal syntax). Here, to access the phone charge for room three, you would use:

```
costs[2].phone;
```

INDEX NUMBER:

VALUE:

0	:	{accom: 420, food: 40, phone: 10}
1	:	{accom: 460, food: 20, phone: 20}
2	:	{accom: 230, food: 0, phone: 0}
3	:	{accom: 620, food: 150, phone: 60}

What are Built -in Objects?

- Browsers come with a set of built-in objects that represent things like the **browser window** and the **current web page** shown in that window.
- These built-in objects act like a **toolkit** for creating interactive web pages.
- The objects you create will usually be specifically written to suit ***your needs***.
 - *They model the data used within, or contain functionality needed by, your script.*
- Whereas, the built-in objects contain functionality commonly needed by **many scripts**.

What are Built -in Objects?

- As soon as a web page has loaded into the browser, these objects are available to use in your scripts.
- These built-in objects help you get a wide range of information such as:
 - **width** of the browser window
 - **content** of the main heading in the page
 - **length** of text a user entered into a form field.
- Access their properties or methods using **dot notation**.

What are Built-in Objects?

- Three groups of Built-in Objects:

- 1 **Browser Object Model**

- contains objects that represent the **current browser window or tab**.
- contains objects that model things like **browser history** and the **device's screen**.

- 2 **Document Object Model**

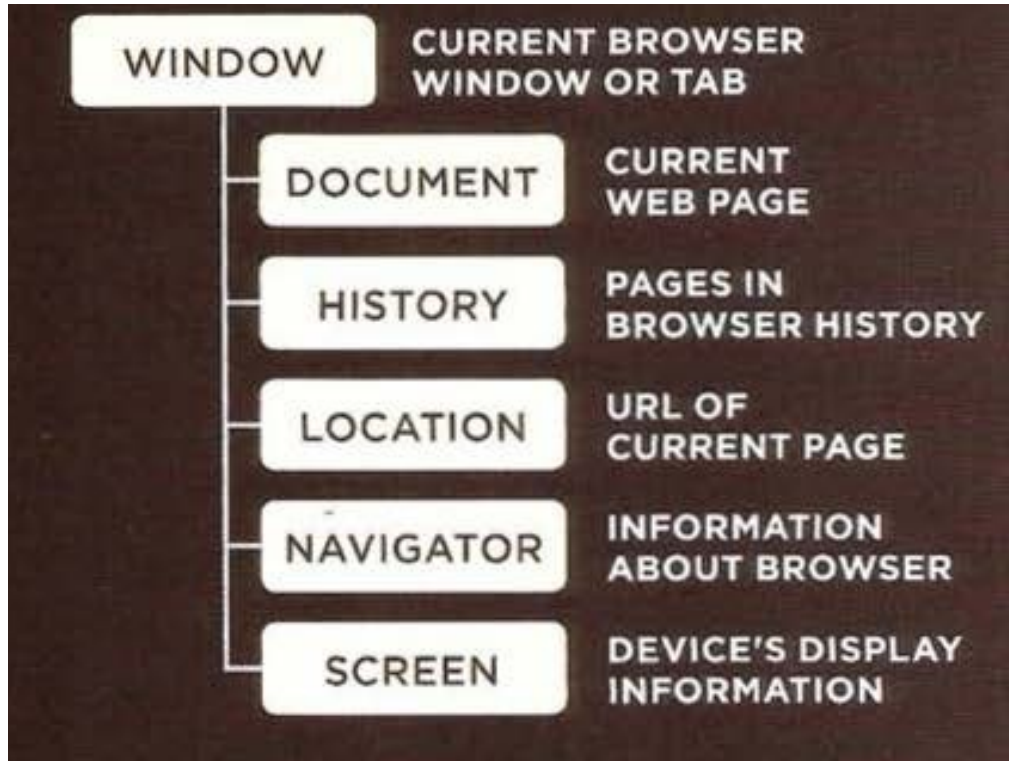
- Uses objects to create a **representation of the current page**.
- It creates a **new object for each element** (and each individual section of text) within the page.

- 3 **Global JavaScript Objects**

- represent things that the JavaScript language needs to create a **model of**.
- **For example**, there is an object that deals only with dates and times.

Browser Object Model

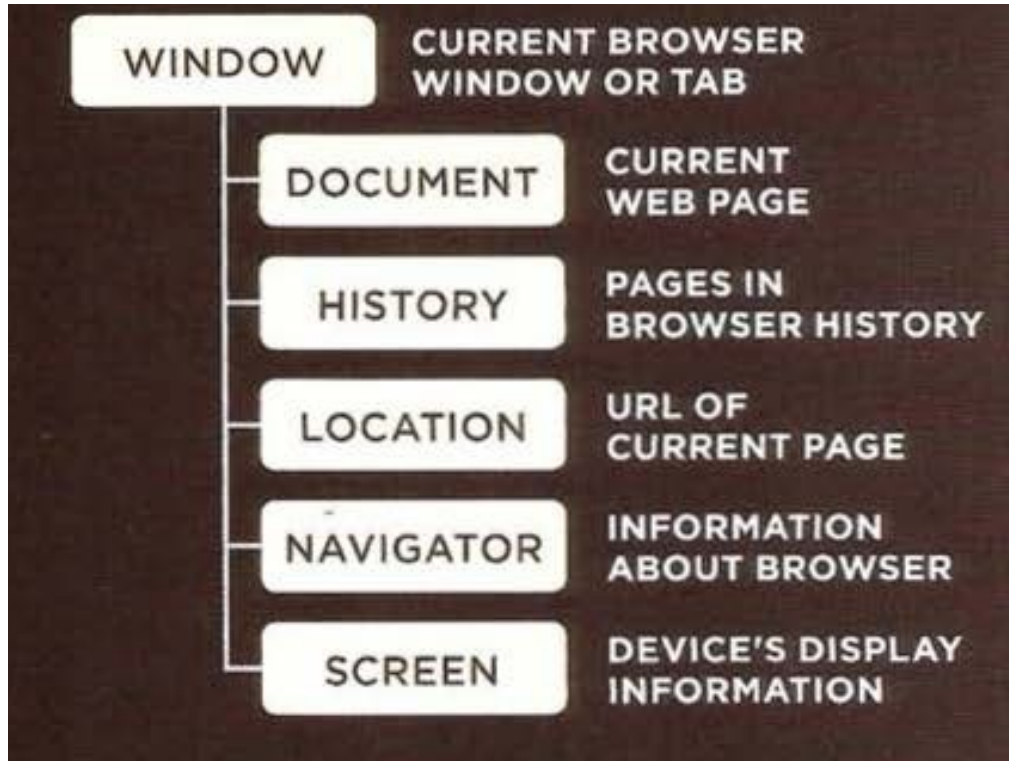
- **BOM** creates a model of the browser tab or window.



- The topmost object is the **window** object, which represents current browser window or tab.
- Its child objects represent other browser features.

Browser Object Model

- **BOM** creates a model of the browser tab or window.



- Example

→ ***window.print();***

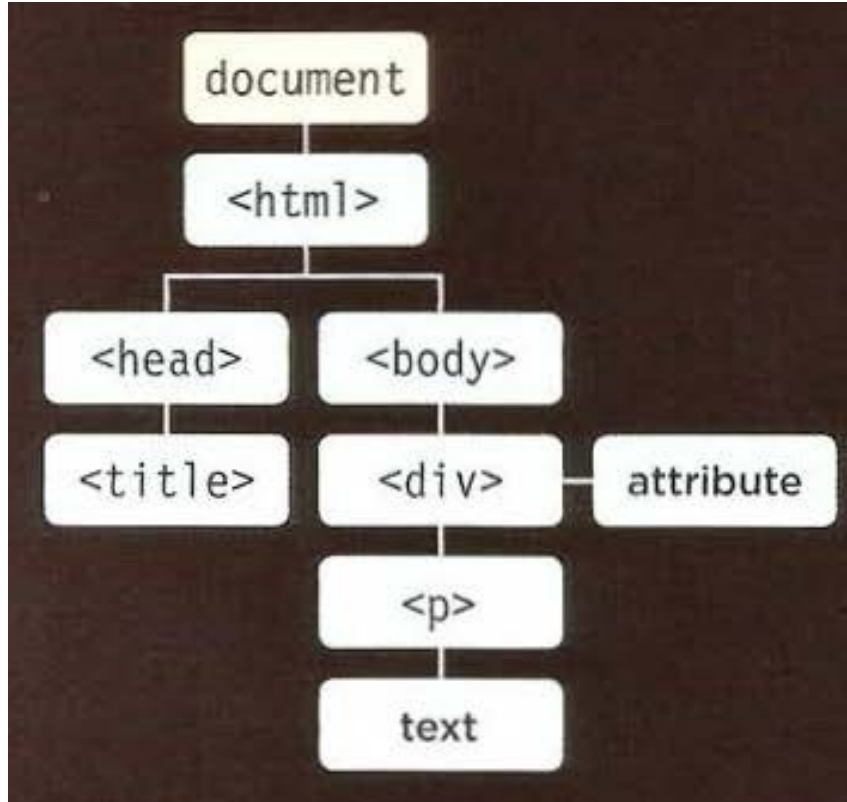
Browser's print dialog box will be shown.

→ ***window.screen.width;***

width property will let you find the width of the device's screen in pixels

Document Object Model

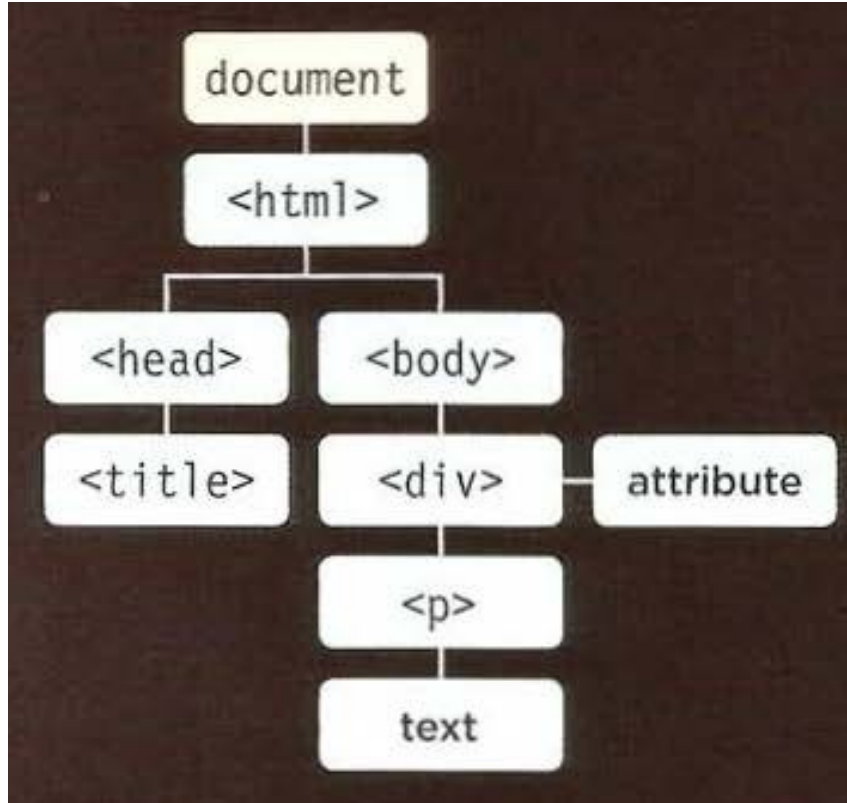
- **DOM** creates a model of the current web page.



- The topmost object is the **document** object, which represents the page as a whole.
- Its child objects represent other items on the page.

Document Object Model

- **DOM** creates a model of the current web page.



- Example

→ **`document.getElementById("id");`**

*Gets an element by the value of its **id** attribute.*

→ **`document.lastModified;`**

lastModified property will tell you the date that the page was last updated.

Global JavaScript Objects

- The global objects do not form a single model.
 - *A group of individual objects that relate to different parts of the JavaScript language.*

These objects represent basic data types:

STRING

FOR WORKING WITH STRING
VALUES

NUMBER

FOR WORKING WITH NUMERIC
VALUES

BOOLEAN

FOR WORKING WITH BOOLEAN
VALUES

These objects help deal with real-world concepts:

DATE

TO REPRESENT AND HANDLE
DATES

MATH

FOR WORKING WITH NUMBERS
AND CALCULATIONS

REGEX

FOR MATCHING PATTERNS
WITHIN STRINGS OF TEXT

Global JavaScript Objects

- Example

→ The **String** object's **toUpperCase()** method makes all letters in the following variable uppercase:

name.toUpperCase();

→ The **Maths** object's **PI** property will return the value of pi:

Math.PI;

The Browser Object Model: The Window Object

- The **window** object represents the current browser window or tab.
 - *topmost object in the Browser Object Model, and it contains other objects that tell you about the browser.*

Property	Description
window . innerHeight	Height of window (excluding browser chrome/user interface) (in pixels)
window.innerWidth	Width of window (excluding browser chrome/user interface) (in pixels)
window.pageXOffset	Distance document has been scrolled horizontally (in pixels)
window . pageYOffset	Distance document has been scrolled vertically (in pixels)
window.screenX	X-coordinate of pointer, relative to top left corner of screen (in pixels)

The Browser Object Model: The Window Object

Property	Description
window . screenY	Y-coordinate of pointer, relative to top left corner of screen (in pixels)
window.location	Current URL of window object (or local file path)
window.document	Reference to document object, which is used to represent the current page contained in window
window.history	Reference to history object for browser window or tab, which contains details of the pages that have been viewed in that window or tab
window. history . length	Number of items in history object for browser window or tab
window.screen	Reference to screen object
window.screen . width	Accesses screen object and finds value of its width property (in pixels)
window. screen.height	Accesses screen object and finds value of its height property (in pixels)

The Browser Object Model: The Window Object

Method	Description
<code>window . alert ()</code>	Creates dialog box with message (user must click OK button to close it)
<code>window. open ()</code>	Opens new browser window with URL specified as parameter (if browser has pop-up blocking software installed, this method may not work).
<code>window.print()</code>	Tells browser that user wants to print contents of current page (acts like user has clicked a print option in the browser's user interface)

Using the Browser Object Model

JAVASCRIPT

c03/js/window-object.js

```
① [var msg = '<h2>browser window</h2><p>width: ' + window.innerWidth + '</p>';  
   msg += '<p>height: ' + window.innerHeight + '</p>';  
   msg += '<h2>history</h2><p>items: ' + window.history.length + '</p>';  
② [msg += '<h2>screen</h2><p>width: ' + window.screen.width + '</p>';  
   msg += '<p>height: ' + window.screen.height + '</p>';  
③ [var el = document.getElementById('info');  
   el.innerHTML = msg;  
④ alert('Current page: ' + window.location);
```

The Document Object Model: The Document Object

- The topmost object in the Document Object Model (or DOM) is the document object.
 - *It represents the web page loaded into the current browser window or tab.*

Property	Description
document.title	Title of current document
document.lastModified	Date on which document was last modified
document . URL	Returns string containing URL of current document
document.domain	Returns domain of current document

The Document Object Model: The Document Object

- The topmost object in the Document Object Model (or DOM) is the document object.
 - *It represents the web page loaded into the current browser window or tab.*

Method	Description
document.write()	Writes text to document
document.getElementById()	Returns element, if there is an element with the value of the id attribute that matches
document.querySelectorAll()	Returns list of elements that match a CSS selector, which is specified as a parameter
document.createElement()	Creates new element
document.createTextNode()	Creates new text node

Using the Document Object

JAVASCRIPT

c03/js/document-object.js

```
① [var msg = '<p><b>page title: </b>' + document.title + '<br />';  
   msg += '<b>page address: </b>' + document.URL + '<br />';  
   msg += '<b>last modified: </b>' + document.lastModified + '</p>';  
  
② [var el = document.getElementById('footer');  
   el.innerHTML = msg;
```

Global Objects: String Objects

- Whenever you have a value that is a string, you can use the properties and methods of the **String** object on that value.

PROPERTY	DESCRIPTION
<code>length</code>	Returns number of characters in the string in most cases (see note bottom-left)

METHOD	DESCRIPTION
<code>toUpperCase()</code>	Changes string to uppercase characters
<code>toLowerCase()</code>	Changes string to lowercase characters
<code>charAt()</code>	Takes an index number as a parameter, and returns the character found at that position
<code>indexOf()</code>	Returns index number of the first time a character or set of characters is found within the string

Global Objects: String Objects

<code>lastIndexOf()</code>	Returns index number of the last time a character or set of characters is found within the string
----------------------------	---

<code>substring()</code>	Returns characters found between two index numbers where the character for the first index number is included and the character for the last index number is not included
--------------------------	---

<code>split()</code>	When a character is specified, it splits the string each time it is found, then stores each individual part in an array
----------------------	---

<code>trim()</code>	Removes whitespace from start and end of string
---------------------	---

<code>replace()</code>	Like find and replace, it takes one value that should be found, and another to replace it (by default, it only replaces the first match it finds)
------------------------	---

Global Objects: String Objects

H o m e s w e e t h o m e

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

EXAMPLE

RESULT

`saying.length;`

H o m e s w e e t h o m e

16

EXAMPLE

RESULT

`saying.toUpperCase();`

H o m e s w e e t h o m e

'HOME SWEET HOME '

`saying.toLowerCase();`

H o m e s w e e t h o m e

'home sweet home '

`saying.charAt(12);`

H o m e s w e e t h o m e

'o'

`saying.indexOf('ee');`

H o m e s w e e t h o m e

7

Global Objects: String Objects

```
.....
```

```
saying.lastIndexOf('e');  H o m e   s w e e t   h o m e   14
```

```
.....
```

```
saying.substring(8,14);  H o m e   s w e e t   h o m e   'et hom'
```

```
.....
```

```
saying.split(' ');      H o m e   s w e e t   h o m e   ['Home', 'sweet', 'home', '']
```

```
.....
```

```
saying.trim();          H o m e   s w e e t   h o m e   'Home sweet home'
```

```
.....
```

```
saying.replace('me','w'); H o m e   s w e e t   h o m e   'How sweet home '
```


Working with Strings

JAVASCRIPT

c03/js/string-object.js

```
① var saying = 'Home sweet home';  
② var msg = '<h2>length</h2><p>' + saying.length + '</p>';  
    msg += '<h2>uppercase</h2><p>' + saying.toUpperCase() + '</p>';  
    msg += '<h2>lowercase</h2><p>' + saying.toLowerCase() + '</p>';  
    msg += '<h2>character index: 12</h2><p>' + saying.charAt(12) + '</p>';  
③ msg += '<h2>first ee</h2><p>' + saying.indexOf('ee') + '</p>';  
    msg += '<h2>last e</h2><p>' + saying.lastIndexOf('e') + '</p>';  
    msg += '<h2>character index: 8-14</h2><p>' + saying.substring(8, 14) + '</p>';  
    msg += '<h2>replace</h2><p>' + saying.replace('me', 'w') + '</p>';  
  
④ [var el = document.getElementById('info');  
    el.innerHTML = msg;
```

Global Objects: Number Objects

- Whenever you have a value that is a number, you can use the methods and properties of the **Number** object on it.

METHOD	DESCRIPTION
<code>isNaN()</code>	Checks if the value is not a number
<code>toFixed()</code>	Rounds to specified number of decimal places (returns a string)
<code>toPrecision()</code>	Rounds to total number of places (returns a string)
<code>toExponential()</code>	Returns a string representing the number in exponential notation

Working with Decimal Numbers

JAVASCRIPT

c03/js/number-object.js

① `var originalNumber = 10.23456;`

3 decimal places

`var msg = '<h2>original number</h2><p>' + originalNumber + '</p>';`

② `msg += '<h2>toFixed()</h2><p>' + originalNumber.toFixed(3); + '</p>';`

③ `msg += '<h2>toFixed()</h2><p>' + originalNumber.toFixed(3) + '</p>';`

`var el = document.getElementById('info');`

`el.innerHTML = msg;` *3 digits*

Global Objects: Math Object

- The Math object has properties and methods for mathematical constants and functions.

PROPERTY	DESCRIPTION
<code>Math.PI</code>	Returns pi (approximately 3.14159265359)

METHOD	DESCRIPTION
<code>Math.round()</code>	Rounds number to the nearest integer
<code>Math.sqrt(<i>n</i>)</code>	Returns square root of positive number, e.g., <code>Math.sqrt(9)</code> returns 3
<code>Math.ceil()</code>	Rounds number up to the nearest integer
<code>Math.floor()</code>	Rounds number down to the nearest integer
<code>Math.random()</code>	Generates a random number between 0 (inclusive) and 1 (not inclusive)

Math Object to create Random Numbers

JAVASCRIPT

c03/.

```
var randomNum = Math.floor((Math.random() * 10) + 1);  
  
var el = document.getElementById('info');  
el.innerHTML = '<h2>random number</h2><p>' + randomNum + '</p>';
```

Creating an instance of the Date Object



Global Objects: Date Object (and Time)

- Once you have created a **Date** object, the following methods let you set and retrieve the time and date that it represents.

METHOD		DESCRIPTION
<code>getDate()</code>	<code>setDate()</code>	Returns / sets the day of the month (1-31)
<code>getDay()</code>		Returns the day of the week (0-6)
<code>getFullYear()</code>	<code>setFullYear()</code>	Returns / sets the year (4 digits)
<code>getHours()</code>	<code>setHours()</code>	Returns / sets the hour (0-23)
<code>getMilliseconds()</code>	<code>setMilliseconds()</code>	Returns / sets the milliseconds (0-999)
<code>getMinutes()</code>	<code>setMinutes()</code>	Returns / sets the minutes (0-59)

Global Objects: Date Object (and Time)

<code>getMonth()</code>	<code>setMonth()</code>	Returns / sets the month (0-11)
<code>getSeconds()</code>	<code>setSeconds()</code>	Returns / sets the seconds (0-59)
<code>getTime()</code>	<code>setTime()</code>	Number of milliseconds since January 1, 1970, 00:00:00 UTC (Coordinated Universal Time) and a negative number for any time before
<code>getTimezoneOffset()</code>		Returns time zone offset in mins for locale
<code>toString()</code>		Returns "date" as a human-readable string
<code>toTimeString()</code>		Returns "time" as a human-readable string
<code>toISOString()</code>		Returns a string representing the specified date

Creating a Date Object

JAVASCRIPT

```
① var today = new Date();  
② var year = today.getFullYear();  
③ [var el = document.getElementById('footer');  
   [el.innerHTML = '<p>Copyright &copy;' + year + '</p>';
```


Working with Dates & Times

To specify a date and time, you can use this format:

YYYY, MM, DD, HH, MM, SS
1996, 04, 16, 15, 45, 55

This represents 3:45pm and 55 seconds on April 16, 1996.

The order and syntax for this is:

Year	four digits
Month	0-11 (Jan is 0)
Day	1-31
Hour	0-23
Minutes	0-59
Seconds	0-59
Milliseconds	0-999

Working with Dates & Times

Another way to format the date and time is like this:

MMM DD, YYYY HH:MM:SS

Apr 16, 1996 15:45:55

You can omit the time portion if you do not need it.

Working with Dates & Times

JAVASCRIPT

c03/js/date-object-difference

```
① var today = new Date();  
   var year = today.getFullYear();  
   var est = new Date('Apr 16, 1996 15:45:55');  
② var difference = today.getTime() - est.getTime();  
③ difference = (difference / 31556900000);  
  
var elMsg = document.getElementById('message');  
elMsg.textContent = Math.floor(difference) + ' years of online travel advice';
```

References

- 1 JavaScript and JQuery – Interactive Front-end Web Development, (Jon Duckett), John Wiley and Sons, Inc.
- 2 <https://developer.mozilla.org/en-US/docs/Web/JavaScript>