

Back-end Web Development

February 25th, 2020



Dorset College Dublin

JavaScript

What we covered last week...

- **Functions**

- Optional parameters
- Default parameters
- Variable-Length
Argument lists
- Rest parameters
- Rest operator

- **Arrays**

- Creating an array via
array literal and
constructor

- Array holes
- Adding and deleting
array elements
- Iterating arrays
- Length
- Const in arrays
- Reading and writing
array elements
- Sparse arrays

JavaScript

What we will cover today...

- **Objects**
 - key/value
 - Literal notation
 - constructor notation
 - Accessing an object and dot notation
 - Updating an object
 - Many objects
- Object instances
- Create and access constructor notation
- Adding and removing properties
- .this

JS Objects

Objects **group together** a set of variables and functions to create a **model**. Variables and names in an object take on a few names.

- If a variable is part of an object, it is called a **property** of that object
- Properties are the **characteristics** of an object
- If a function is part of an object, it is called a **method**
- Methods represent the **tasks** that are associated with an object

— — —

Objects

```
var hotel = {  
  name: 'Clayton',  
  rooms: 110,  
  booked: 25,  
  gym: true,  
  roomTypes: ['double', 'twin', 'suite', 'king'],  
  checkAvailability: function() {  
    return this.rooms - this.booked;  
  }  
};
```

key

value

- Properties and methods have a **name** and a **value**, the name is called a **key**
- Value can be a string, number, boolean, array or another object
- The value of a method is always a **function**

Creating an Object: Literal Notation

Literal notation is the easiest and most popular way to create objects.

- The object is stored in a variable called `hotel` which you refer to as the **hotel object**
- Separate each key from it's value using a **colon**
- Separate each property and method with a **comma** (but not after the last value)
- The **this** keyword is used to **access** the objects properties

```
var hotel = {  
  name: 'Clayton',  
  rooms: 110,  
  booked: 25,  
  gym: true,  
  roomTypes: ['double', 'twin', 'suite', 'king'],  
  checkAvailability: function() {  
    return this.rooms - this.booked;  
  }  
};
```

Properties

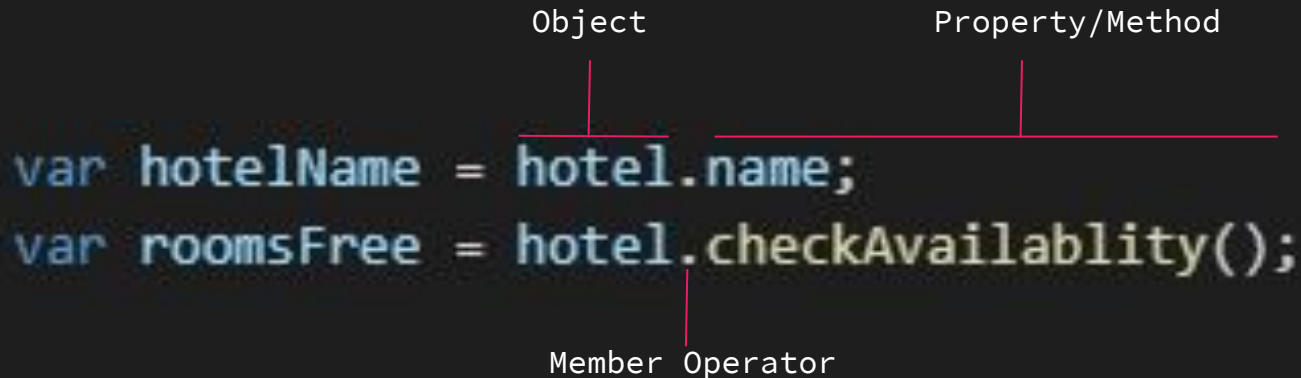
method

Accessing an Object and Dot Notation

- You use the **dot notation** to access properties or methods of an object
- You can also access properties using **square brackets**

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

- To access the property of an object you use the name of the object, followed by a period and then the name of the property/method you want to access. This is known as a dot notation
- The property/method on the right is the **member** of the object on the left



The diagram illustrates the components of dot notation using two code examples. The first example is `var hotelName = hotel.name;` and the second is `var roomsFree = hotel.checkAvailability();`. Red lines and labels identify the parts: 'Object' points to 'hotel' in both; 'Property/Method' points to 'name' and 'checkAvailability()'; 'Member' points to 'hotel' in the second example; and 'Operator' points to the dot '.' in the second example.

```
var hotelName = hotel.name;
```

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

Object

Property/Method

Member Operator

Creating an Object: Literal Notation

- The object is called a hotel which represents a hotel called “Clayton”
- Content is **updated** with the data from this object
- The name is displayed accessing the **name property** and the number of vacant rooms displayed using the **checkAvailability()** method
- There can be **one hotel object** with different properties
- The only thing that changes in an object is the **properties**, so let's say we have 1000 hotels, we do not need to recreate a hotel object, just change its properties

```
var hotel = {  
  // Object properties  
  name: 'Clayton',  
  rooms: 110,  
  booked: 25,  
  gym: true,  
  roomTypes: ['double', 'twin', 'suite', 'king'],  
  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**
- You can then add the properties and methods to the object
- Each statement that adds a property or method should end with a **semicolon**
- To create an empty object using literal notation use **var hotel = {}**

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

```
hotel.name = 'Calyton';  
hotel.rooms = 45;  
hotel.booked = 24;
```

Properties

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

method

Updating an Object

- To **update the value** of properties, use the dot notation or square brackets
- They work on objects created using a literal notation or a constructor notation
- Use the **delete keyword** to delete a property
- If an object doesn't have the property you are trying to update, it will be added to the object
- You can also update using the **bracket syntax**
- To clear the value of a property, just set it to a **blank string**

Property Name Property Value

object

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

Member Operator Assignment Operator

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

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

```
delete hotel.name;
```

Creating Many Objects: Constructor Notation

- Object constructors can use a **function as a template** for creating objects
- The `Hotel` function on the right is used as a template that **represents the hotel** which has properties and methods
- There are **3 parameters** that set the value of a property in the object
- The **this** keyword is used instead of the object name to indicate that it **belongs** to this function
- The name of a constructor function usually begins with a **capital letter** (unlike other functions)

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

Creating Many Objects: Constructor Notation

```
var clyatonHotel = new Hotel('Clayton', 128, 25);  
var lemonHotel = new Hotel('Lemon', 76, 14)
```

- You 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
- In the example above there are 2 hotel objects which represent the Hotel object
- When the new keyword calls the constructor function, it creates a new object
- Both objects automatically get the same method defined in the constructor function

Create & Access Constructor Notation



TRAVELWORTHY



hotel availability

CLAYTON

85

rooms left

other hotels

Clayton rooms: 65
Lemon rooms: 41

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

```
var clayHotel = new Hotel('Clayton', 68, 3);  
var lemonHotel = new Hotel('Lemon', 56, 15);
```

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

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

Adding and Removing Properties

- Once you created an object using a literal or a constructor you can add new properties
- Use the dot notation to **add properties**
- In the example on the right we are creating an **instance** of the hotel object using an object literal
- We give the hotel object 2 extra properties gym and pool
- Having these properties added to the object, you can **access them** like any other properties of the object
- To delete a property just use the **delete keyword** followed by the object name and dot notation

```
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 = hotel.pool;  
  
var elGym = document.getElementById('gym');  
elGym.className = hotel.gym;
```

Recap

Create the object, then add properties and methods

Literal Notation

Empty object is created with properties and methods followed.

```
var hotel = {}  
  
hotel.name = 'Clayton';  
hotel.rooms = 110;  
hotel.booked = 25;  
✓ hotel.checkAvailablity = function() {  
  |   return this.rooms - this.booked;  
  |  
};
```

Object Constructor Notation

Once an empty object is created, the syntax for adding/removing properties is the same.

```
var hotel = new Object();  
  
hotel.name = 'Calyton';  
hotel.rooms = 45;  
hotel.booked = 24;  
hotel.checkAvailablity = function(){  
  |   return this.rooms - this.booked;  
  |  
};
```


Creating an object with properties and methods

Literal Notation

A colon separates the key/value pairs. There is a comma between each one.

```
var hotel = {  
  // Object properties  
  name: 'Clayton',  
  rooms: 110,  
  booked: 25,  
  gym: true,  
  roomTypes: ['double', 'twin', 'suite', 'king'],  
  checkAvailability: function() {  
    return this.rooms - this.booked;  
  }  
};
```

Object Constructor Notation

A function can be used to create multiple objects. The `this` keyword is used instead of 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 clyatonHotel = new Hotel('Clayton', 128, 25);  
var lemonHotel = new Hotel('Lemon', 76, 14 );
```

.this

This keyword is commonly used **inside functions** and **objects**. Depending on where the function is declared, the meaning of this changes. Always refers to **one object**.

- A function declared at the **top level** of a script (not inside an object or a function) is in the **global scope/context**
- The **window** object is a default object, if we use this inside a function in the global context it refers to the **window object**
- The **this** keyword is a **reference** to the object that the function is created inside

```
function windowSize(){  
    var width = this.innerWidth;  
    var height = this.innerHeight;  
    return [height, width];  
}
```

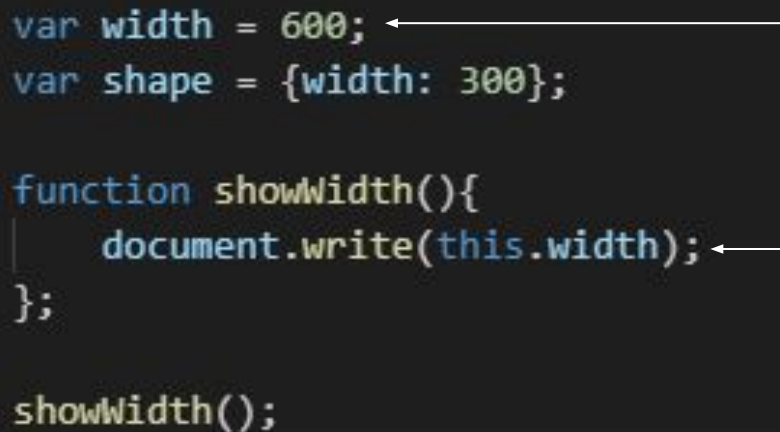
this... Global Variables

- All global variables also become **properties** of the window object
- When a function is in the global context, you can access global variables using the **window object** as well as its other properties
- The example on the right has the **showWidth()** function in the global scope and **this.width** refers to the width variable
- The function would write the value of **600** onto the page using the document object's **write()** method

```
var width = 600;
var shape = {width: 300};

function showWidth(){
    document.write(this.width);
};

showWidth();
```



this... A method of an Object

- When a function is defined inside an object, it becomes a **method**
- In a method this refers to the **containing object**
- The example of the right shows the getArea() method appearing inside the shape object
- In this example this refers to the shape object it is **contained in**
- It would be the same as **writing return shape.width * shape.height;**

```
var shape = {  
  width: 600,  
  height: 400,  
  getArea: function(){  
    return this.width * this.height;  
  }  
};
```

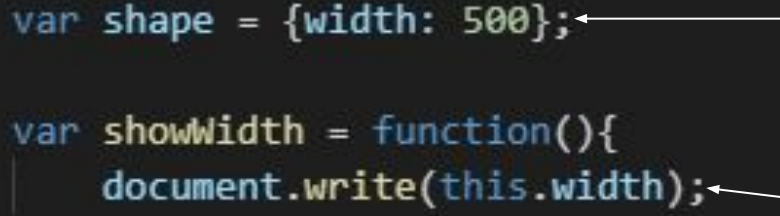
this... Function Expression as a method

- If a named function has been defined in a global scope it is then used as a method of an object, **this** then refers to the object it is contained within
- The example on the right uses the same showWidth() function expression as the one in the previous slide, it is however assigned as a method of an object
- The last line indicates that showWidth() is used as a method of the shape object

```
var width = 600;
var shape = {width: 500};

var showWidth = function(){
    document.write(this.width);
};

shape.getWidth = showWidth;
shape.getWidth();
```



Lets Code!

Your lab for today is to use JS objects and functions integrated with HTML and CSS.

Take the example code provided in class and expand from it.
Eg. add any extra information such as whether there is gym in the hotel, is breakfast included etc.

