ECMAScript 6

JS

FUNCTIONS, METHODS & OBJECTS

Functions let you group a series of statements together to perform a specific task.

Functions are reusable and save you from writing out the same code over and over.

DECLARING A FUNCTION

Declaration - Expression

```
// funcion declaration
function dizerOla() {
    resultado = 'Oi da minha função';
    resultado += '<br />Bem vindo ao JS...'
}
dizerOla(); //Hoisting, can invoke before declaration



// funcion expression
let dizerOla = function(){
    resultado = 'Oi da minha função';
    resultado += '<br />Bem vindo ao JS...';
}
dizerOla(); // No Hoisting can´t invoke without declaration first
```

Parameters

```
// 4 - Functions with parameters
function dizerOla(nome){
    resultado = `Olá ${nome} bem vindo ao JS`;
}
dizerOla('joao');


// optional parameters
// ES5:
function dizerOla(nome){
    nome = nome || 'Johnny';
    resultado += `<br />Olá ${nome} bem vindo ao JS`;
}
dizerOla();

//ES6
function dizerOla(nome='Joao Silva'){
    resultado += `<br />Olá ${nome} bem vindo ao JS`;
}
dizerOla();
dizerOla('Maria');
```

Rest operator

```
// REST operatoras argument
 function dizerOla(...nomes){
     for(nome of nomes){
      resultado += `<br />Olá ${nome} bem vindo ao JS`;
     }
 }
 dizerOla('Joao', 'Manuel', 'MAria');
```

Return values

```
//functions returning values
function calcularArea(largura, altura){
   let area = largura * altura;
   return area;
}
resultado = `A área de um restangulo de 5 por 4 é ${calcularArea(5,4)}`;
```

Arrow Functions

```
// 8 - Arrow Functions
let ola = () => resultado='Ola da minha arrow function';
ola();

let calcularArea = (alt,larg) => (alt*larg);

let calcularArea = (alt,larg) => {
   let area = alt*larg;
   return area;
};
resultado = `A área de um restangulo de 5 por 4 é ${calcularArea(5,4)}`;
```

IIFE & Modules

```javascript
///Imediate Invoke Functions Expressions (IIFE)
(function(){
   console.log('ola da minh aIFFE.....')
}());

/// Javscript Modules (closures) with IIFE
let meuMundo = (function(){
   let nome = 'Joao';
   let apelido = 'Silva';

   let ola = () => `Olá ${nome} ${apelido}`;

   console.log(ola());

   return {
    //ola:ola,
    ola,
   }
}())
meuMundo.ola();
```

ES6 array methods

```
// map() arrow function
let duplos = numeros.map( n => n*2 );
resultado += `<h3>Duplos: ${duplos}</h3>`;


// filter()
let pares = numeros.filter( n => n%2 === 0);
resultado += `<h3>Pares: ${pares}</h3>`;


// reduce()
numeros = [1,2,3,4,5];
//ac=0
let soma = numeros.reduce( (ac,cv) => (ac+cv) );
resultado += `<h3>Soma: ${soma}</h3>`;

// others: find() , findIndex(), from(), some(), every(),...
```

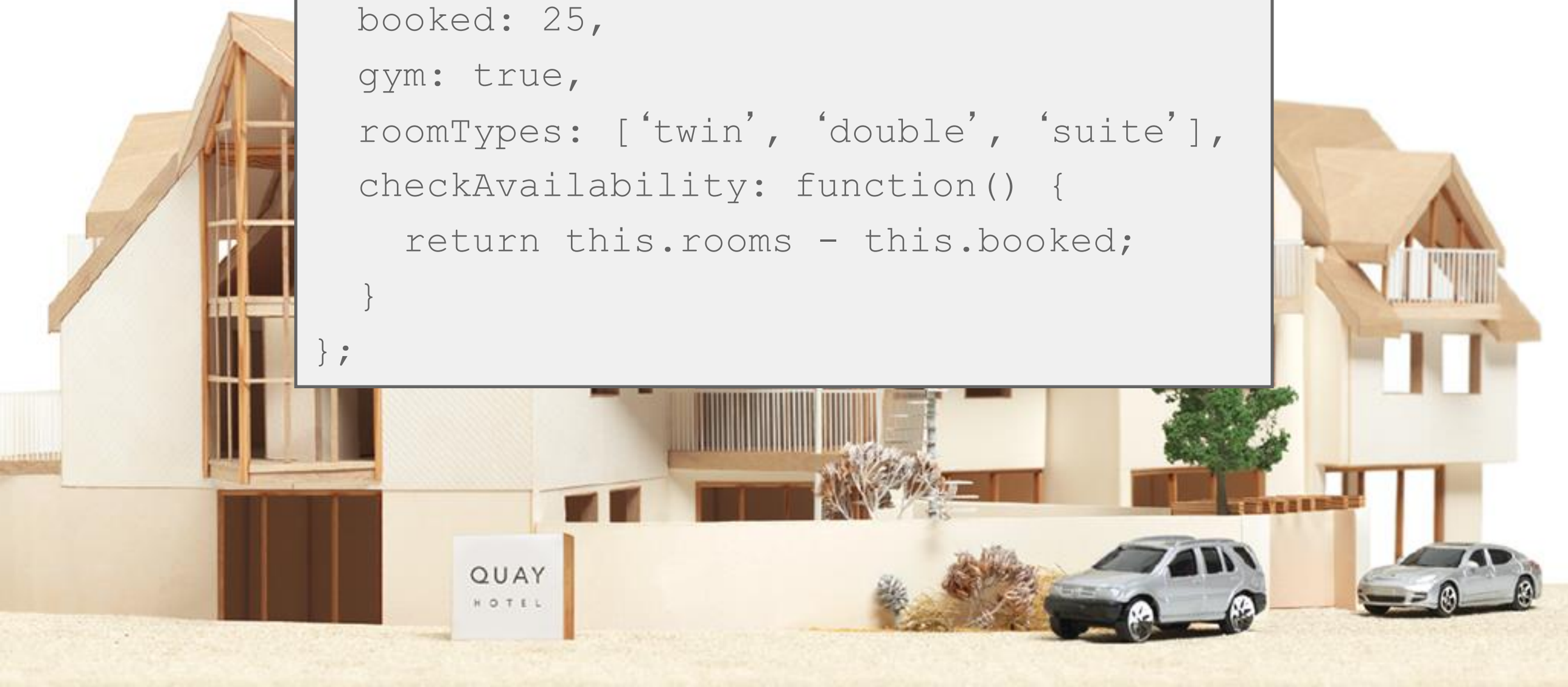
OBJECTS

Objects **group together** variables and functions to create a model.

In an object, variables and functions take on new names. They become **properties** and **methods**.

Each property or method consists of a **name** (also known as a **key**) and its corresponding **value**.

```
var hotel = {
  name: ‘Quay’,
  rooms: 40,
  booked: 25,
  gym: true,
  roomTypes: [‘twin’, ‘double’, ‘suite’],
  checkAvailability: function() {
    return this.rooms - this.booked;
  }
};
```



```
var hotel = {
  name: ‘Quay’,
  rooms: 40,
  booked: 25,
  gym: true,
  roomTypes: [‘twin’, ‘double’, ‘suite’],
  checkAvailability: function() {
    return this.rooms - this.booked;
  }
};
```

NAMES (KEYS)

```
var hotel = {
  name: ‘Quay’,
  rooms: 40,
  booked: 25,
  gym: true,
  roomTypes: [‘twin’, ‘double’, ‘suite’],
  checkAvailability: function() {
    return this.rooms - this.booked;
  }
};
```

VALUES

```
var hotel = {
  name: ‘Quay’,
  rooms: 40,
  booked: 25,
  gym: true,
  roomTypes: [‘twin’, ‘double’, ‘suite’],
  checkAvailability: function() {
    return this.rooms - this.booked;
  }
};
```

PROPERTIES

```
var hotel = {
  name: ‘Quay’,
  rooms: 40,
  booked: 25,
  gym: true,
  roomTypes: [‘twin’, ‘double’, ‘suite’],
  checkAvailability: function() {
    return this.rooms - this.booked;
  }
};
```

METHOD

```
var hotel = {
  name: ‘Quay’,
  rooms: 40,
  booked: 25,
  gym: true,
  roomTypes: [‘twin’, ‘double’, ‘suite’],
  checkAvailability: function() {
    return this.rooms - this.booked;
  }
};
```

ACCESSING OBJECTS

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

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

OBJECT

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

PROPERTY

```
var hotelName = hotel[‘name’];
```

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

OBJECT

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

PROPERTY

UPDATING OBJECTS

```
hotel.name = ‘Park’;
```

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

OBJECT

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

PROPERTY

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

NEW VALUE

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

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

OBJECT

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

PROPERTY

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

NEW VALUE

BUILT-IN OBJECTS

1

1

Browser Object Model

THE WEB BROWSER

BROWSER OBJECT MODEL

PROPERTIES INCLUDE:

```
window.innerHeight

window.innerWidth

window.screenX

window.screenY
```

METHODS INCLUDE:

```
window.print()
```

1

Browser Object Model

THE WEB BROWSER

1

Browser Object Model

THE WEB BROWSER

2

Document Object Model

THE PAGE LOADED IN THE WEB BROWSER (OR TAB)

DOCUMENT OBJECT MODEL

PROPERTIES INCLUDE:

```
document.title

document.lastModified
```

METHODS INCLUDE:

```
document.write()

document.getElementById()
```

1

Browser Object Model

THE WEB BROWSER

2

Document Object Model

THE PAGE LOADED IN THE WEB BROWSER (OR TAB)

3

1

Browser Object Model

THE WEB BROWSER

2

Document Object Model

THE PAGE LOADED IN THE WEB BROWSER (OR TAB)

3

Global JavaScript Objects

GENERAL PURPOSE OBJECTS JAVASCRIPT NEEDS TO WORK

GLOBAL JAVASCRIPT OBJECTS

PROPERTIES INCLUDE:

```
saying.length
```

METHODS INCLUDE:

```
saying.toUpperCase()
saying.toLowerCase()
saying.charAt(3)
```

EXEMPLOS....

Scope & Context

Javascript

context and scope
are not the same

every function invocation has both a scope and a context associated with it

scope is function-based

context is object-based

scope pertains to the variable access of a function when it is invoked

context is always the value of the ”THIS” keyword

which is a reference to the object that “owns” the currently executing code

Variable Scope

variable can be defined in either local or global scope

What is “this” Context

context is most often determined by how a function is invoked

when a function is called as a method of an object, this is set to the object the method is called on

```
let obj = {
    foo: function() {
        return this;
    }
};

obj.foo() === obj; // true
```

when called as an unbound function, this will default to the global context or window object in the browser

```
function foo() {
    alert(this);
}

foo() // window or undefined is strict mode
```

JavaScript is a single threaded language

When the JavaScript interpreter initially executes code, it first enters into a global execution context

Each invocation of a function from this point on will result in the creation of a new execution context

For each execution context there is a scope chain coupled with it

```
function first() {
    second();
      function second() {
          third();
            function third() {
                fourth();
                  function fourth() {
                      // do something
                  }
            }
      }
}
first();
```

Closures

```
function foo() {
    var localVariable = 'private variable';
    return function() {
        return localVariable;
    }
}

var getLocalVariable = foo();
getLocalVariable() // "private variable"
```

Modules (pattern)

```
var Module = (function() {
    var privateProperty = 'foo';

    function privateMethod(args) {
        // do something
    }

    return {

        publicProperty: '',

        publicMethod: function(args) {
            // do something
        },

        privilegedMethod: function(args) {
            return privateMethod(args);
        }
    };
})();
```

call(), apply() e bind()

```
function user(firstName, lastName, age) {
    // do something
}

user.call(window, 'John', 'Doe', 30);
user.apply(window, ['John', 'Doe', 30]);
```

Exemplos