

# PeopleCert

## Software Development Skills

C# Stream

Lesson 27

Trainer Presentation



# Copyright

The contents of this workshop are protected by copyright and can be reproduced under the Terms of Use agreed between PeopleCert and the ATO using this material only.

Material in this presentation has been sourced from the bibliography listed in the certification's Syllabus.

No part of this document may be reproduced in any form without the written permission of PeopleCert International Ltd. Permission can be requested at [www.peoplecert.org](http://www.peoplecert.org).

e-mail: [info@peoplecert.org](mailto:info@peoplecert.org), [www.peoplecert.org](http://www.peoplecert.org)

## **Copyright © 2017-2019 PeopleCert International Ltd.**

All rights reserved. No part of this publication may be reproduced or transmitted in any form and by any means (electronic, photocopying, recording or otherwise) except as permitted in writing by PeopleCert International Ltd. Enquiries for permission to reproduce, transmit or use for any purpose this material should be directed to the publisher.

## **DISCLAIMER**

This publication is designed to provide helpful information to the reader. Although every care has been taken by PeopleCert International Ltd in the preparation of this publication, no representation or warranty (express or implied) is given by PeopleCert International Ltd. as publisher with respect as to the completeness, accuracy, reliability, suitability or availability of the information contained within it and neither shall PeopleCert International Ltd be responsible or liable for any loss or damage whatsoever (indicatively but not limited to, special, indirect, consequential) arising or resulting of virtue of information, instructions or advice contained within this publication.)

- Document Version: PC-TRAINER\_SD 1.0 | February 2020

**PeopleCert**

All talents, certified.

# Coding Bootcamp

## LESSON 28

### Web Design and Development Fundamentals (Front End)

---

- 
- |                              |                |  |
|------------------------------|----------------|--|
| ▪ Introduction to JavaScript | ▪ Booleans     | ▪ 4.6.1 Code in JavaScript, variables, functions             |
| ▪ Client-Side Scripting      | ▪ Operators    | ▪ 4.6.2 Code using advance java script: if conditions, loops |
| ▪ Language Syntax            | ▪ Conditionals |  |
| ▪ Variables                  | ▪ Loops        |  |
| ▪ Strings                    | ▪ Arrays       |  |
| ▪ Numbers                    | ▪ Objects      |  |



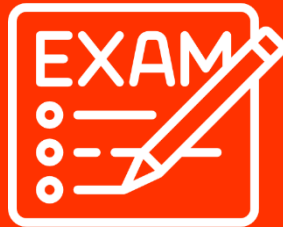
Start and  
end times



Participation  
is essential!



Lunch and  
breaks



Exam



Feedback



# Syllabus

Category	Topic	Task
<b>FSD_4 Web Design and Developme nt Fundament als (Front- End)</b>	<b>4.6 JavaScript/ jQuery</b>	4.6.1 Code in JavaScript, variables, functions
		4.6.2 Code using advance java script: if conditions, loops
		4.6.3 Improve usability of forms, validate data from the user



## Contents | Learning Objectives

- ✓ Introduction to JavaScript
- ✓ Client-Side Scripting
- ✓ Language Syntax
- ✓ Variables
- ✓ Strings
- ✓ Numbers
- ✓ Booleans
- ✓ Operators
- ✓ Conditionals
- ✓ Loops
- ✓ Arrays
- ✓ Objects
- ✓ Learn the basics of JavaScript
- ✓ Familiarize yourself with the syntax of JavaScript
- ✓ Understand the basic usage of JavaScript
- ✓ Practice on the fundamental concepts and features of the language
- ✓ Code in JavaScript using variables, functions
- ✓ Learn how arrays are used in JavaScript
- ✓ Understand conditionals, loops
- ✓ Familiarize with JavaScript Objects



## JavaScript (Basic) Part II

# Objects in JavaScript

- JavaScript is an object-based language based on prototypes instead of classes.

## Definition

- An object is a collection of properties, and a property is an association between a name (or key) and a value. A property's value can be a function, in which case the property is known as a method.

```
// "Object" is a prototype, "myCar" is an object
var myCar = new Object();

// myCar.brand is a property
myCar.brand = "Ford";

// myCar.getBrand is a method
myCar.getBrand = function () {
  return this.brand; // "this" refers to the current object: myCar
}
```

- JavaScript is not a full-fledged object-oriented programming language
- It does not have classes per se, and it does not support many of the patterns you'd expect from an object-oriented language like inheritance and polymorphism
- The language does, however, support objects
- Objects in JavaScript, just as in many other programming languages, can be compared to objects in real life.
- An object is a standalone entity, with properties and type.
- Compare it with a cup, for example. A cup is an object, with properties. A cup has a color, a design, weight, a material it is made of, etc. The same way, JavaScript objects can have properties, which define their characteristics.
- Objects can have constructors, properties, and methods associated with them
- There are objects that are included in the JavaScript language
- You can also define your own kind of objects

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>



# Objects and Properties

- A JavaScript object has properties associated with it.
- A property of an object can be explained as a variable that is attached to the object.
- Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object.
- You access the properties of an object with a simple dot-notation:

**ObjectName.propertyName**

- Or using the bracket notation: **ObjectName['propertyName']**
- Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value.
- An object property name can be any valid JavaScript string, or anything that can be converted to a string, including the empty string.
- Any property name that is not a valid JavaScript identifier (for example, a property name that has a space or a hyphen, or that starts with a number) can only be accessed using the square bracket notation.
- This notation is also very useful when property names are to be dynamically determined (when the property name is not determined until runtime).

```
// four variables are created and assigned in a single go,
// separated by commas
var myObj = new Object(),
    str = 'myString',
    rand = Math.random(),
    obj = new Object();

myObj.type           = 'Dot syntax';
myObj['date created'] = 'String with space';
myObj[str]           = 'String value';
myObj[rand]          = 'Random Number';
myObj[obj]           = 'Object';
myObj['']            = 'Even an empty string';

console.log(myObj);
```

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>

## Create New Objects

- JavaScript has a number of predefined objects.
- You can create your own objects.
- Create objects through:
  - Using an object initializer.
  - Using a constructor function and then instantiate an object invoking that function in conjunction with the new operator.
  - Using the **objects.create** method

### Object Creation

```
var objName = {  
    name1: value1,  
    name2: value2,  
    // ...  
    nameN: valueN  
};
```

**Note: Unassigned properties of an object are undefined (and not null).**

// first create an empty object

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

// then define properties for this object

```
objName.name1 = value1;
```

```
objName.name2 = value2;
```



Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>

## Using Object Initializers

- The syntax for an object using an object initializer is:
 

```
var obj = { property_1: value_1,    // property_# may be an identifier...
           2: value_2,             // or a number...
           // ...,
           'property n': value_n }; // or a string
```
- where obj is the name of the new object, each property\_i is an identifier (either a name, a number, or a string literal), and each value\_i is an expression whose value is assigned to the property\_i.
- The obj and assignment is optional; if you do not need to refer to this object elsewhere, you do not need to assign it to a variable. (Note that you may need to wrap the object literal in parentheses if the object appears where a statement is expected, so as not to have the literal be confused with a block statement.)

### Examples

- The following statement creates an object and assigns it to the variable x if and only if the expression cond is true:

```
if (cond) var x = {greeting: 'hi there'};
```

- The following example creates myHonda with three properties. Note that the engine property is also an object with its own properties.


```
var myHonda = {color: 'red', wheels: 4, engine: {cylinders: 4, size: 2.2}};
```

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>



## Class Exercise/Discussion #1a

- Create an object named myCar and give it properties named make, model, and year (use your favorite car)



## Nested objects

- **Nested objects** are the objects that are inside another object.
- In the following example 'vehicles' is an object which is inside a main object called 'person'. Using dot notation the nested objects' property(car) is accessed.

```
var person = {  
  name:"Ram",  
  age:27,  
  vehicles: {  
    car:"ford focus",  
    bike:"ktm-duke",  
    boat:"triton"  
  }  
}  
  
alert("Car: " + person.vehicles.car);
```

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>

## Create Object via Function Constructors

- You can create an object with these two steps:
  1. Define the object type by writing a constructor function. There is a strong convention, with good reason, to use a capital initial letter.
  2. Create an instance of the object with new.
- To define an object type, create a function for the object type that specifies its name, properties, and methods.

```
function MyObject( name ){
  this.name = name; // Initialize an object property named name with the argument passed to the function
}
```

```
const anObject = new MyObject( "toy" );
// anObject is now a object with the following structure:
{ name: "toy" }
```

### Constructors

- Normally to create a new object we use the new keyword, the class name, and ( ) brackets with n optional parameters inside, comma delimited as follows:
 

```
var someObject = new ObjectName(p1,p2,..., pn);
```
- For some classes, shortcut constructors are defined
 

```
var greeting = "Good Morning";
```
- vs the formal:
 

```
var greeting = new String("Good Morning");
```

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>



## Class Exercise/Discussion

### Class Exercise/Discussion #10b

- Suppose you want to create an object type for cars. You want this type of object to be called car, and you want it to have properties for make, model, and year. Use a constructor approach.

### Class Exercise/Discussion #10c

- Suppose you want to create an object type for cars. You want this type of object to be called car, and you want it to have properties for make, model, and year. Use a Class constructor approach.



## Browse Objects | for loop

- When you want to browse objects you can use for in

```
var txt = "";  
var person = { fname: "John", lname: "Doe", age: 27 };  
for (var x in person) {  
    txt +=person[x] + " " ;  
}
```

## Properties & Property Enumeration

- Each object might have properties that can be accessed, depending on its definition
- When a property exists, it can be accessed using dot notation where a dot between the instance name and the property references that property  
*//show someObject.property to the user*  
`alert(someObject.property);`

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>





## Enumerate the Properties of an Object

There are three native ways to list/traverse object properties:

- **for...in loops** : This method traverses all enumerable properties of an object and its prototype chain
- **Object.keys(o)** : This method returns an array with all the own (not in the prototype chain) enumerable properties' names ("keys") of an object o.
- **Object.getOwnPropertyNames(o)** : This method returns an array containing all own properties' names (enumerable or not) of an object o.

### Example

```
function listAllProperties(o) {  
    var objectToInspect;  
    var result = [];  
    for(objectToInspect = o; objectToInspect !== null; objectToInspect =  
Object.getPrototypeOf(objectToInspect)) {  
        result = result.concat(Object.getOwnPropertyNames(objectToInspect));  
    }  
    return result;  
}
```

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>



## Methods

- Objects can also have methods, which are functions associated with an instance of an object
- These methods are called using the same dot notation as for properties, but instead of accessing a variable, we are calling a method

`someObject.doSomething();`

- Methods may produce different output depending on the object they are associated with because they can utilize the internal properties of the object

- **Example**

```
let Person = {  
  name: "John",  
  sayHi: function() {  
    alert("Hi buddy!");  
  }  
};
```

```
Person.sayHi();
```

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>

## Using the Object.create method

- Objects can also be created using the Object.create() method.
- This method can be very useful, because it allows you to choose the prototype object for the object you want to create, without having to define a constructor function

- **Example**

```
const person = {
  isHuman: false,
  printIntroduction: function() {
    console.log(`My name is ${this.name}. Am I human? ${this.isHuman}`);
  }
};
```

```
const me = Object.create(person);
me.name = 'Matthew'; // "name" is a property set on "me", but not on "person"
me.isHuman = true; // inherited properties can be overwritten
me.printIntroduction(); // expected output: "My
name is M
atthew. Am I human? true"
```

### Using the Object.Create method **Example**

```
// Animal properties and method encapsulation
var Animal = {
  type: 'Invertebrates', // Default value of properties
  displayType: function() { // Method which will display type of Animal
    console.log(this.type);
  }
};

// Create new animal type called animal1
var animal1 = Object.create(Animal);
animal1.displayType(); // Output: Invertebrates

// Create new animal type called Fishes
var fish = Object.create(Animal);
fish.type = 'Fishes';
fish.displayType(); // Output: Fishes
```

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>

## Adding a Method

- Adding a method to a javascript object is easier than adding a method to an object constructor. We need to assign the method to the existing property to ensure task completion.
- In the following example, initially, the object type is created and later on, the properties of the object were created. Once the creation of properties is done, a method is assigned to each of the objects and the properties were accessed using the method as our requirement.
- **Example**

```
function Business(name, property, age) {
    this.Name = name;
    this.prop = property;
    this.age = age;
}
var person1 = new Business("John", "100.000", "73");
var person2 = new Business("Joe", "250.000", "54");
person1.det = function() {
    return this.Name + " "+" has a property of net worth "+ "" + this.prop;
};
person2.det = function() {
    return this.Name + " "+" has a property of net worth "+ "" + this.prop;
};
alert(person2.det() +" and "+person1.det());;
```

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>

# Prototypes

- Prototypes are the mechanism by which JavaScript objects inherit features from one another
- JavaScript is often described as a prototype-based language — to provide inheritance, objects can have a prototype object, which acts as a template object that it inherits methods and properties from.
- An object's prototype object may also have a prototype object, which it inherits methods and properties from, and so on. This is often referred to as a prototype chain, and explains why different objects have properties and methods defined on other objects available to them.
- The properties and methods are defined on the prototype property on the Objects' constructor functions, not the object instances themselves.
- In JavaScript, a link is made between the object instance and its prototype (its `__proto__` property, which is derived from the prototype property on the constructor), and the properties and methods are found by walking up the chain of prototypes.
- Suppose we have defined a constructor function `Person`
- We have then created an object instance like this:  

```
let person1 = new Person(...);
```
- We will see the members defined on `person1`'s constructor — `Person()`. We will however also see some other members — `toString`, `valueOf`, etc — these are defined on `Person()`'s constructor's prototype object, which is `Object`.



Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>



## Class-based vs. prototype-based languages

- Class-based object-oriented languages, such as Java and C++, are founded on the concept of two distinct entities: **classes** and **instances**.
  - A **class** defines all of the properties (considering methods and fields in Java, or members in C++, to be properties) that characterize a certain set of objects. A class is an abstract thing, rather than any particular member of the set of objects it describes. For example, the Employee class could represent the set of all employees.
  - An **instance**, on the other hand, is the instantiation of a class; that is, one of its members. For example, Victoria could be an instance of the Employee class, representing a particular individual as an employee. An instance has exactly the same properties of its parent class (no more, no less).
- A **prototype-based language**, such as **JavaScript**, does not make this distinction: it simply has **objects**.
- A prototype-based language has the notion of a prototypical object, an object used as a template from which to get the initial properties for a new object.
- Any object can specify its own properties, either when you create it or at run time. In addition, any object can be associated as the prototype for another object, allowing the second object to share the first object's properties.

Source: <https://docs.oracle.com/cd/E19957-01/816-6409-10/obj2.htm>

## Comparison of class-based (Java) and prototype-based (JavaScript) object systems

Class-based (Java)	Prototype-based (JavaScript)
Class and instance are distinct entities.	All objects can inherit from another object.
Define a class with a class definition; instantiate a class with constructor methods.	Define and create a set of objects with constructor functions.
Create a single object with the new operator.	Same.
Construct an object hierarchy by using class definitions to define subclasses of existing classes.	Construct an object hierarchy by assigning an object as the prototype associated with a constructor function.
Inherit properties by following the class chain.	Inherit properties by following the prototype chain.
Class definition specifies all properties of all instances of a class. Cannot add properties dynamically at run time.	Constructor function or prototype specifies an initial set of properties. Can add or remove properties dynamically to individual objects or to the entire set of objects.

Source: <https://docs.oracle.com/cd/E19957-01/816-6409-10/obj2.htm>

## Using Prototypes to Extend Other Objects

```
String.prototype.countChars = function (c) {  
  var count=0;  
  for (var i=0;i<this.length;i++) {  
    if (this.charAt(i) == c)  
      count++;  
  }  
  return count;  
}  
var msg = "Hello World";  
console.log(msg + "has" + msg.countChars("l") + " letter l's");
```

### Modifying Prototypes

- Modifying the prototype property of a constructor function — methods added to the prototype are then available on all object instances created from the constructor.
- The following code adds a new method to the constructor's prototype property:

```
Person.prototype.farewell = function() {  
  alert(this.name + ' has left the building. Bye for now!');  
};  
person1.farewell();
```

Source: [https://developer.mozilla.org/en-US/docs/Learn/Tools\\_and\\_testing/Cross\\_browser\\_testing/JavaScript](https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/JavaScript)





## Class Exercise/Discussion #2

- Suppose you want to create an object type for Bank Account. You want this type of object to be called BankAccount, and you want it to have properties for owner name, balance and two functions deposit and withdraw.
- Create a BankAccount Object
- Create BankAccount object using the two constructors approach.



## Objects Included in JavaScript

- A number of useful objects are included with JavaScript including:
  - Array
  - Boolean
  - Date
  - Math
  - String
  - Dom objects
  - RegExp
  - Function

### Math

- The Math class allows one to access common mathematic functions and common values quickly in one place
- The Math object contains static methods such as `max()`, `min()`, `pow()`, `sqrt()`, and `exp()`, and trigonometric functions such as `sin()`, `cos()`, and `arctan()`
- Many mathematical constants are defined such as PI, E, SQRT2, and some others
  - `Math.PI; // 3.141592657`
  - `Math.sqrt(4); // square root of 4 is 2.`
  - `Math.random(); // random number between 0 and 1`

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>



## Date

- The Date class is yet another helpful included object you should be aware of
- It allows you to quickly calculate the current date or create date objects for particular dates
- To display today's date as a string, we would simply create a new object and use the toString() method

```
var d = new Date();  
// This outputs Today is Mon Nov 12 2012 15:40:19 GMT-0700  
alert ("Today is " + d.toString());
```
- There are methods to access the year, month and so on from the Date object:

```
// This output is Number of Month 10  
alert("Month is " + d.getMonth());
```

## Window

- The window object in JavaScript corresponds to the browser itself
- Through it, you can access the current page's URL, the browser's history, and what's being displayed in the status bar, as well as opening new browser windows
- In fact, the alert() function mentioned earlier is actually a method of the window object

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>

## Regular Expressions – RegExp(1)

- A regular expression (also “regexp”, or just “reg”) consists of a pattern and optional flags.
- There are two syntaxes that can be used to create a regular expression object.
  - `regexp = new RegExp("pattern", "flags");`
  - `regexp = /pattern/gmi; // with flags g,m and i`

- **Example**

```
let str = "We will, we will rock you";
```

```
alert( str.match(/we/gi) ); // We,we (an array of 2 substrings that match)
```

- A regular expression consists of a pattern and optional flags: g, i, m, u, s, y.
- Without flags and special symbols (that we’ll study later), the search by a regexp is the same as a substring search.
- The method `str.match(regexp)` looks for matches: all of them if there’s g flag, otherwise, only the first one.
- The method `str.replace(regexp, replacement)` replaces matches found using regexp with replacement: all of them if there’s g flag, otherwise only the first one.
- The method `regexp.test(str)` returns true if there’s at least one match, otherwise, it returns false.

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>



## Regular Expressions **Examples**

```
let str = "I love JavaScript";  
let regexp = /LOVE/i;  
alert( regexp.test(str) ); // true  
  
alert( "We will, we will".replace(/we/ig, "I") ); // I will, I will  
  
let matches = "JavaScript".match(/HTML/) || [];  
if (!matches.length) {  
    alert("No matches"); // no matches  
}
```



# Map

- Map is a collection of keyed data items, just like an Object. But the main difference is that Map allows keys of any type.
- Methods and properties are:
  - `new Map()` – creates the map.
  - `map.set(key, value)` – stores the value by the key.
  - `map.get(key)` – returns the value by the key, undefined if key doesn't exist in map.
  - `map.has(key)` – returns true if the key exists, false otherwise.
  - `map.delete(key)` – removes the value by the key.
  - `map.clear()` – removes everything from the map.
  - `map.size` – returns the current element count.

## Map Example

```
let map = new Map();map.set('1', 'str1'); // a string key
map.set(1, 'num1'); // a numeric key
map.set(true, 'bool1'); // a boolean key
```

```
// Map keeps the type, so these two are different:
alert( map.get(1) ); // 'num1'
alert( map.get('1') ); // 'str1'
alert( map.size ); // 3
```

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>



## Set

- A Set is a special type collection – “set of values” (without keys), where each value may occur only once.
- Its main methods are:
  - `new Set(iterable)` – creates the set, and if an iterable object is provided (usually an array), copies values from it into the set.
  - `set.add(value)` – adds a value, returns the set itself.
  - `set.delete(value)` – removes the value, returns true if value existed at the moment of the call, otherwise false.
  - `set.has(value)` – returns true if the value exists in the set, otherwise false.
  - `set.clear()` – removes everything from the set.
  - `set.size` – is the elements count.
- The main feature is that repeated calls of `set.add(value)` with the same value don’t do anything. That’s the reason why each value appears in a Set only once.
- For example, we have visitors coming, and we’d like to remember everyone. But repeated visits should not lead to duplicates. A visitor must be “counted” only once.

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>



## Set **Example**

```
let set = new Set();
let john = { name: "John" };
let pete = { name: "Pete" };
let mary = { name: "Mary" };

// visits, some users come multiple times
set.add(john);
set.add(pete);
set.add(mary);
set.add(john);
set.add(mary);

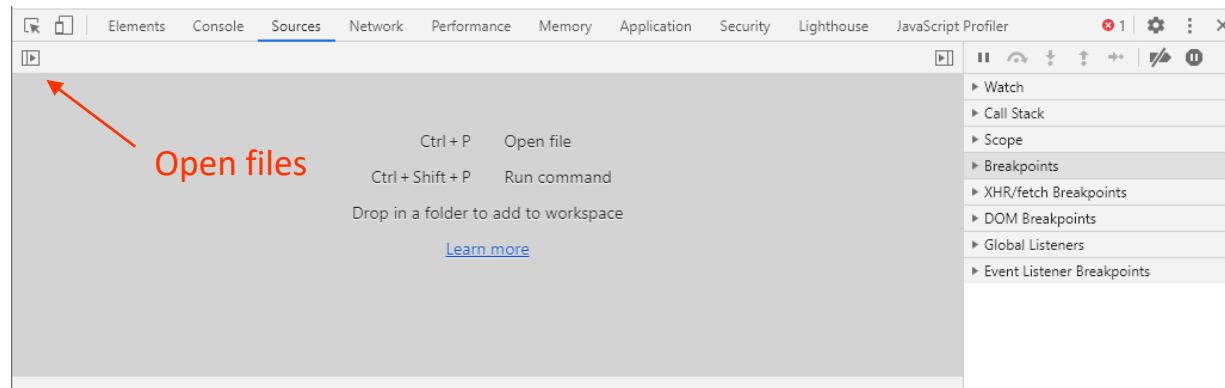
// set keeps only unique values
alert( set.size ); // 3

for (let user of set) {
  alert(user.name); // John (then Pete and Mary)
}
```



## Debugging in Chrome(1)

- Open your page in Chrome.
- Turn on developer tools with F12 (Mac: Cmd+Opt+I).
- Select the Sources pane



- The above button opens the tab with files.
- Click it and select your js in the tree view.

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>

## Debugging in Chrome(2)

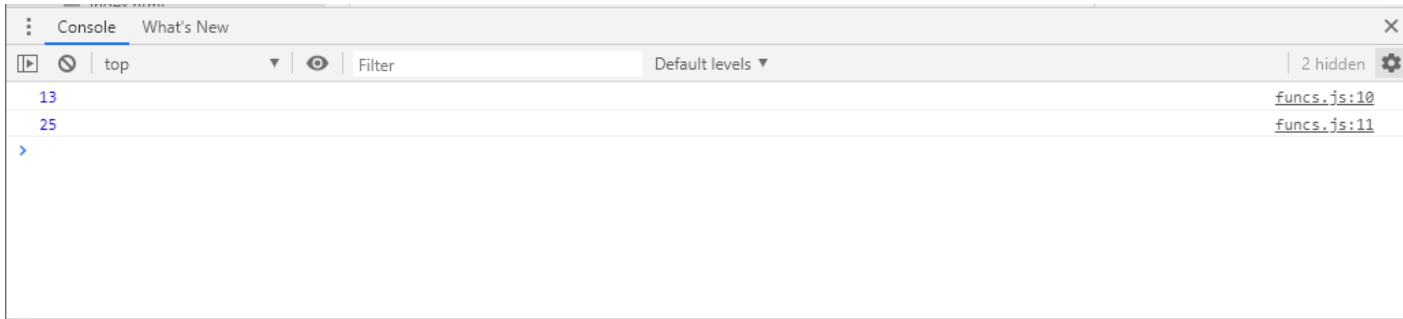


- The Sources panel has 3 parts:
  - The **File Navigator** pane lists HTML, JavaScript, CSS and other files, including images that are attached to the page. Chrome extensions may appear here too.
  - The **Code Editor** pane shows the source code.
  - The **JavaScript Debugging** pane is for debugging, we'll explore it soon.

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>

## Debugging in Chrome(3)

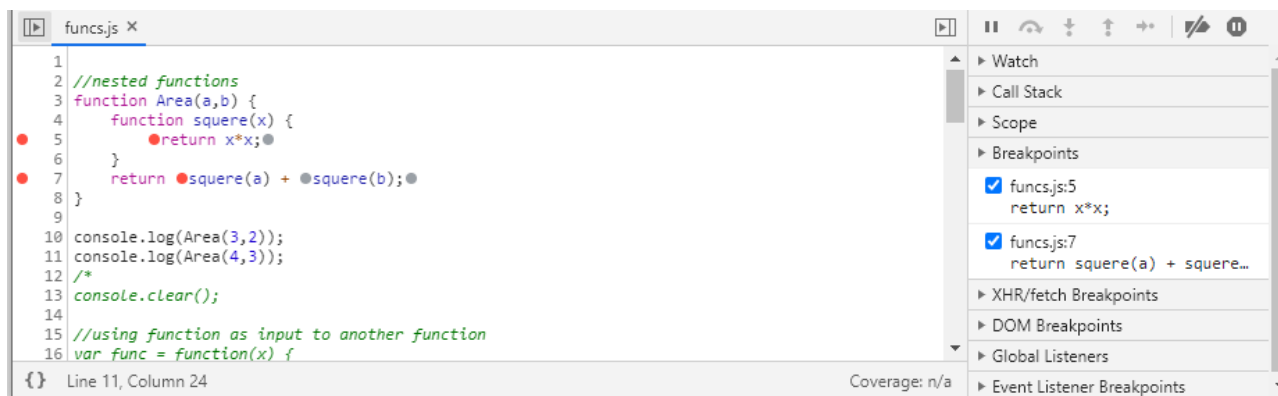
- If we press Esc, then a console opens below. We can type commands there and press Enter to execute.
- After a statement is executed, its result is shown below



Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>

## Debugging in Chrome(4)

- Breakpoints
- Let's examine what's going on within the code of the example page. In js, click at line number 4.
- Also click on the number for line 7.
- It should look like this:

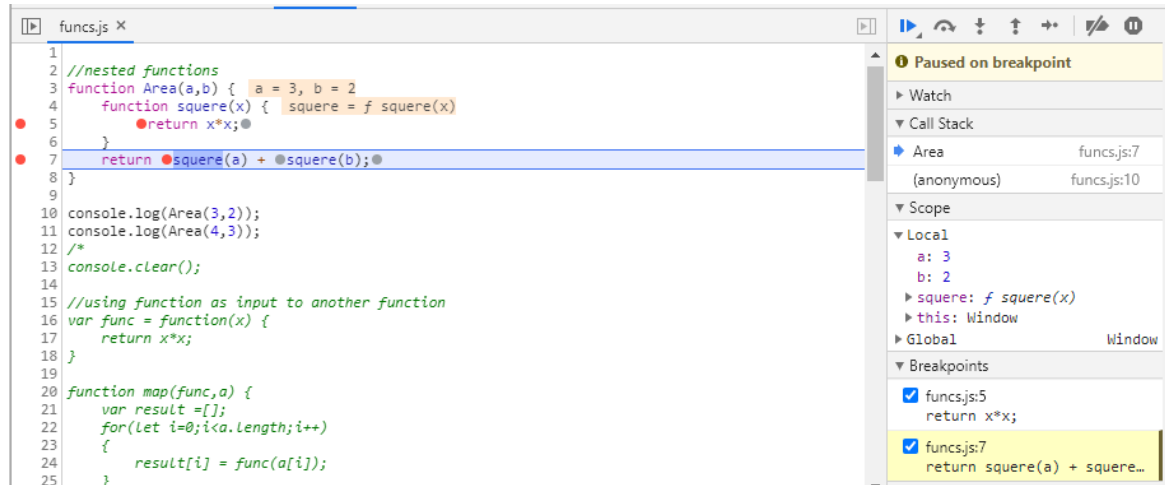


- A breakpoint is a point of code where the debugger will automatically pause the JavaScript execution.
- While the code is paused, we can examine current variables, execute commands in the console etc. In other words, we can debug it.
- We can always find a list of breakpoints in the right panel. That's useful when we have many breakpoints in various files. It allows us to:
  - Quickly jump to the breakpoint in the code (by clicking on it in the right panel).
  - Temporarily disable the breakpoint by unchecking it.
  - Remove the breakpoint by right-clicking and selecting Remove.
  - ...And so on.

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>

## Debugging in Chrome(5)

- Press F5 (Windows, Linux) or Cmd+R (Mac).
- As the breakpoint is set, the execution pauses at the 5th line:



Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>




## Debugging in Chrome(6)

- Watch – shows current values for any expressions.
  - You can click the plus + and input an expression. The debugger will show its value at any moment, automatically recalculating it in the process of execution.
- Call Stack – shows the nested calls chain.
  - At the current moment the debugger is inside hello() call, called by a script in index.html (no function there, so it's called "anonymous").
  - If you click on a stack item (e.g. "anonymous"), the debugger jumps to the corresponding code, and all its variables can be examined as well.
- Scope – current variables.
  - Local shows local function variables. You can also see their values highlighted right over the source.
  - Global has global variables (out of any functions).
  - There's also this keyword there that we didn't study yet, but we'll do that soon.

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>

## Debugging in Chrome(7)

-  Resumes the execution. If there are no additional breakpoints, then the execution just continues and the debugger loses control.
- The execution has resumed, reached another breakpoint inside js and paused there. Take a look at the “Call Stack” at the right. It has increased by one more call.
- Step: run the next command, hotkey F9
- Step over: run the next command, but don’t go into a function, hotkey F10.
- Step into, hotkey F11.
- Step out: continue the execution till the end of the current function, hotkey Shift+F11.

Sources: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working\\_with\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)  
<https://javascript.info/>  
<https://www.w3schools.com/js/>







## Class Exercise/Discussion #3

- Calculate the difference between 2 dates in days, months and years

## Class Exercise/Discussion #4

- Create an array `arr` and fill it with some values.
- Create a function `unique(arr)` that should return an array with unique items of `arr`.



## Class Exercise/Discussion

### Exercise 1:

- Create an object to hold information on your favorite recipe. It should have properties for title (a string), servings (a number), and ingredients (an array of strings).
- On separate lines (one `console.log` statement for each), log the recipe information so it looks like:
  - Mole
  - Serves: 2
  - Ingredients: cinnamon, cumin, cocoa

### Exercise 2:

Keep track of which books you read and which books you want to read!

- Create an array of objects, where each object describes a book and has properties for the title (a string), author (a string), and `alreadyRead` (a boolean indicating if you read it yet).
- Iterate through the array of books. For each book, log the book title and book author like so: "The Hobbit by J.R.R. Tolkien".
- Now use an `if/else` statement to change the output depending on whether you read it yet or not. If you read it, log a string like 'You already read "The Hobbit" by J.R.R. Tolkien', and if not, log a string like 'You still need to read "The Lord of the Rings" by J.R.R. Tolkien.'



## References / Further Reading

- Randy Connolly and Ricardo Hoar: Fundamentals of Web Development, 2nd Edition
- <https://vanillajstoolkit.com/reference/>
- <https://www.youtube.com/watch?v=W6NZfCO5SIk>
- [https://developer.mozilla.org/en-US/docs/Learn/Getting started with the web/JavaScript basics](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics)
- <https://www.w3schools.com/js/>
- <https://javascript.info/>
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar and types#Variables](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables)
- [developer.mozilla.org/en-US/docs/Web/JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript)
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control flow and error handling](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling)
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops and iteration](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration)
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working with Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)
- <https://cs486t.files.wordpress.com/2019/10/cs486t-ch4-19-20.pdf>
- <https://blog.sessionstack.com/how-javascript-works-service-workers-their-life-cycle-and-use-cases-52b19ad98b58>
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality comparisons and sameness](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness)



## References / Further Reading / Bibliography

Fundamentals of Web Development, 2<sup>nd</sup> Edition

Randy Connolly and Ricardo Hoar

ISBN 9780134481760

© 2018 Pearson



**Any Questions?**



## PeopleCert Values Your Feedback

Like the course?

Have something to say?

Send us your comments at:  
**[academic@peoplecert.org](mailto:academic@peoplecert.org)**

# Thank You!

For latest news and updates follow us.



[linkedin.com/company/peoplecert-group/](https://www.linkedin.com/company/peoplecert-group/)

[twitter.com/peoplecert](https://twitter.com/peoplecert)

[youtube.com/channel/UCLBidKZS9Xk08f\\_PM5Edtfg](https://www.youtube.com/channel/UCLBidKZS9Xk08f_PM5Edtfg)

[facebook.com/peoplecert.org](https://facebook.com/peoplecert.org)