Oggetti

- Gli oggetti sono tipi composti che contengono un certo numero di **proprietà** (attributi)
  - Ogni proprietà ha un **nome** e un **valore**
  - Si accede alle proprietà con l'operatore '**.**' (punto)
  - Le proprietà non sono definite a priori: *possono essere aggiunte dinamicamente*
- Gli oggetti vengono creati usando l'operatore **new**:

  var o = new Object()

- ***Attenzione:*** Object() è un costruttore e non una classe
- Le classi non esistono e quindi i due concetti non si sovrappongono come avviene in Java

Costruire un oggetto

- Un oggetto appena creato è completamente vuoto non ha ne proprietà nè metodi
- Possiamo costruirlo dinamicamente
  - appena assegniamo un valore ad una proprietà la proprietà comincia ad esistere
- Nell'esempio sottostante creiamo un oggetto e gli aggiungiamo 3 proprietà numeriche: x, y e tot:

```
var o = new Object();
o.x = 7;
o.y = 8;
o.tot = o.x + o.y;
alert(o.tot);
```

Costanti oggetto

- Le costanti oggetto (object literal) sono racchiuse fra parentesi graffe **{ }** e contengono un elenco di attributi nella forma: **nome:valore**
  var nomeoggetto = {prop1:val1, prop2:val2, ...}
- Usando le costanti oggetto creiamo un oggetto e le proprietà (valorizzate) nello stesso momento
- I due esempi seguenti sono del tutto equivalenti:

```
var o = new Object();
o.x = 7;
o.y = 8;
o.tot = 15;
alert(o.tot);
```

```
var o = {x:7, y:8, tot:15};
alert(o.tot);
```

Example

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

<p>Creating a JavaScript Object.</p>

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

<script>
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

Example 2

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

<p>Creating and using an object method.</p>

<p>An object method is a function definition, stored as a property
value.</p>

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

<script>
var person = {
    firstName: "John",
    lastName : "Doe",
    id       : 5566,
    fullName : function() {
       return this.firstName + " " + this.lastName;
    }
};

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

Array

- Gli array sono tipi composti i cui elementi sono accessibili mediante un indice numerico
  - l'indice parte da zero (**0**)
  - Non hanno una dimensione prefissata (simili agli ArrayList di Java)
  - Espongono attributi e metodi
- Vengono istanziati con **new Array([*dimensione*])**
- Si possono creare e inizializzare usando delle costanti

array (array literal) delimitate da [ ]:

var *varname* = [*val*,*val2*,...,*valn*]

- Es. **var a = [1,2,3];**
  Possono contenere elementi di tipo eterogeneo:
- Es. **var b = [1,true,"ciao",{x:1,y:2}];**

makePhrase.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>makePhrase</title>
  <script>
    function makePhrases() {
      var words1 = ["24/7", "multi-tier", "30,000 foot", "B-to-B", "win-win"];
      var words2 = ["empowered", "value-added", "oriented", "focused", "aligned"];
      var words3 = ["process", "solution", "tipping-point", "strategy", "vision"];
      var rand1 = Math.floor(Math.random() * words1.length);
      var rand2 = Math.floor(Math.random() * words2.length);
      var rand3 = Math.floor(Math.random() * words3.length);
      var phrase = words1[rand1] + " " + words2[rand2] + " " + words3[rand3];
      alert(phrase);
    }
    makePhrases();
  </script>
</head>
<body></body>
</html>
```

Example

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

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

<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

Example 2

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

<p>The length property returns the length of an array.</p>

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

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.length;
</script>

</body>
</html>
```

Example: delete

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

<p>The delete operator deletes a property from an object.</p>

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

<script>
var person = {
    firstname:"John",
    lastname:"Doe",
    age:50,
    eyecolor:"blue"
};
delete person.age;
document.getElementById("demo").innerHTML =
person.firstname + " is " + person.age + " years old.";
</script>

</body>
</html>
```

Example: in

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

<p>The in operator returns true if the specified property is in the
specified object.</p>

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

<script>
// Arrays
var cars = ["Saab", "Volvo", "BMW"];
// Objects
var person = {firstName:"John", lastName:"Doe", age:50};

document.getElementById("demo").innerHTML =
    ("Saab" in cars) + "<br>" +
    (0 in cars) + "<br>" +
    (1 in cars) + "<br>" +
    (4 in cars) + "<br>" +
    ("length" in cars) + "<br>" +

    ("firstName" in person) + "<br>" +
    ("age" in person) + "<br>" +

    // Predefined objects
    ("PI" in Math) + "<br>" +
    ("NaN" in Number) + "<br>" +
    ("length" in String);
</script>

</body>
</html>
```

Example: instanceof

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

<p>The instanceof operator returns true if the specified object is an
instance of the specified object.</p>

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

<script>
var cars = ["Saab", "Volvo", "BMW"];

document.getElementById("demo").innerHTML =
    (cars instanceof Array) + "<br>" +
    (cars instanceof Object) + "<br>" +
    (cars instanceof String) + "<br>" +
    (cars instanceof Number);
</script>

</body>
</html>
```

Example: void

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


<p>
<a href="javascript:void(0);">
  Useless link
</a>
</p>

<p>
<a href="javascript:void(document.body.style.backgroundColor='red');">
  Click me to change the background color of body to red.
</a>
</p>

</body>
</html>
```

Oggetti e Array

- Gli oggetti in realtà sono **array associativi**
  - strutture composite i cui elementi sono accessibili mediante un indice di tipo stringa (nome) anziché attraverso un indice numerico
- Si può quindi utilizzare anche una sintassi analoga a quella degli array
- Le due sintassi sono del tutto equivalenti e si possono mescolare

```
var o = new Object();
o.x = 7;
o.y = 8;
o.tot = o.x + o.y;
alert(o.tot);
```

```
var o = new Object();
o["x"] = 7;
o.y = 8;
o["tot"] = o.x + o["y"];
alert(o.tot);
```

Stringhe

- Non è facile capire esattamente cosa sono le stringhe in JavaScript
- Potremmo dire che mentre in Java sono oggetti che sembrano dati di tipo primitivo in JavaScript sono **dati di tipo primitivo che sembrano oggetti**
- Sono sequenze arbitrarie di caratteri in formato UNICODE a 16 bit e sono immutabili come in Java
- Esiste la possibilità di definire costanti stringa (string literal) delimitate da apici singoli ('ciao') o doppi ("ciao")
- È possibile la concatenazione con l'operatore **+**
- È possibile la comparazione con gli operatori **< > >= <=** e **!=**

Example

- When comparing two strings, "2" will be greater than "12", because (alphabetically) 1 is less than 2.

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

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

<script>
document.getElementById("demo").innerHTML = "2" < "12";
</script>

</body>
</html>
```

Stringhe come oggetti?

- Possiamo però invocare metodi su una stringa o accedere ai suoi attributi
- Possiamo infatti scrivere
  - var s = “ciao”;
  - var n = s.length;
  - var t = s.charAt(1);
- Non sono però oggetti e la possibilità di trattarli come tali nasce da due caratteristiche:
  - Esiste un tipo wrapper String che è un oggetto
  - *JavaScript fa il boxing in automatico come C#*
    - quando una variabile di tipo valore necessita essere convertita in tipo riferimento, un oggetto box è allocato per mantenere tale valore

String Basics

- You can use double or single quotes

```
var names = ["joe", 'jane', "john", 'juan'];
```

- Strings have length property

```
"Foobar".length → 6
```

- Numbers can be converted to strings
  - Automatic conversion during concatenations.

```
var val = 3 + "abc" + 5;  // result is "3abc5"
```

  - Conversion with fixed precision

```
var n = 123.4567;
var val = n.toFixed(2); // result is 123.46
(not 123.45)
```

String Basics (Continued)

- Strings can be compared with ==

```
"foo" == 'foo' // returns true
```

- Strings can be converted to numbers

```
var i = parseInt("37 blah");
                // result is 37 - ignores blah


var d = parseFloat("6.02 blah");
                // result is 6.02 - ignores blah
```

Core String Methods

- Simple methods
  - Charat, indexof, lastindexof, substring, tolowercase, touppercase

```
"Hello".charAt(1); → "e"
"Hello".indexOf("o"); → 4  // returns -1 if no match
"hello".substring(1,3); → "el"
"Hello".toUpperCase(); → "HELLO"
```

- Methods that use regular expressions
  - match, replace, search, split
- HTML methods
  - anchor, big, bold, fixed, fontcolor, fontsize, italics, link, small, strike, sub, sup

```
"test".bold().italics().fontcolor("red")
→ '<font color="red"><i><b>test</b></i></font>'
```

  - These are technically nonstandard methods, but supported in all major browsers

JavaScript Data Types

```
var length = 16;                                    // Number
var lastName = "Johnson";                           // String
var cars = ["Saab", "Volvo", "BMW"];                // Array
var x = {firstName:"John", lastName:"Doe"};         // Object
```

```
var x;                  // Now x is undefined
var x = 5;              // Now x is a Number
var x = "John";         // Now x is a String
```

```
var carName = "Volvo XC60";   // Using double quotes
var carName = 'Volvo XC60';   // Using single quotes
```

```
var answer = "It's alright";              // Single quote inside double quotes
var answer = "He is called 'Johnny'";     // Single quotes inside double quotes
var answer = 'He is called "Johnny"';     // Double quotes inside single quotes
```

The typeof Operator

- You can use the JavaScript typeof operator to find the type of a JavaScript variable:

```
typeof "John"                 // Returns string
typeof 3.14                   // Returns number
typeof NaN                    // Returns number
typeof false                  // Returns boolean
typeof [1, 2, 3, 4]           // Returns object
typeof {name:'John', age:34}  // Returns object
typeof new Date()             // Returns object
typeof function () {}         // Returns function
typeof myCar                  // Returns undefined (if myCar is not declared)
typeof null                   // Returns object
```