



JavaScript: Un Curso Rápido

Parte II: Funciones y Objetos

Traducción de Dr. Roberto Solís Robles

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Java 6, Ajax, jQuery, GWT, Spring, Hibernate, RESTful Web Services, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Temas

- **Funciones**
 - Fundamenteos
 - Como tipos de datos
 - Funciones anónimas
- **Objetos**
 - Fundamentos
 - La propiedad prototype
 - Espacios de nombres (métodos static)
 - JSON
 - eval
- **Funciones con número variable de argumentos**



Introducción

“JavaScript tiene más en común con los lenguajes funcionales como Lisp o Scheme que con C o Java.”

- Douglas Crockford en “JavaScript: The World’s Most Misunderstood Programming Language”.

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Java 6, Ajax, jQuery, GWT, Spring, Hibernate, RESTful Web Services, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Haciendose Bueno en JavaScript

- **JavaScript no es Java**
 - Si intenta programar JavaScript como Java, *nunca* será bueno en JavaScript.
- **La programación funcional es un enfoque clave**
 - La programación funcional es mucho mas central a la programación en JavaScript que la POO.
 - Los programadores de Java consideran a la programación funcional la parte mas dificil de aprender de JavaScript.
 - Debido a que Java no soporta la programación funcional
 - Programadores que usan Ruby, Lisp, Scheme, Python, ML, Haskell, Clojure, Scala, etc. estan acostumbrados a ella



Funciones

“Es Lisp disfrazado de C.”

- JSON y el guru de YUI Douglas Crockford, describiendo el lenguaje JavaScript en *JavaScript: The Good Parts*.

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Java 6, Ajax, jQuery, GWT, Spring, Hibernate, RESTful Web Services, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Visión General

- **No son similares a Java**
 - Las funciones en JavaScript son *muy* diferentes a los métodos de Java
- **Principales diferencias con Java**
 - Puede tener funciones globales
 - No solamente métodos
 - No puede declarar tipos de retorno o de argumentos
 - El llamador puede proporcionar cualquier numero de argumentos
 - Independientemente de cuantos argumentos haya definido
 - Las funciones son tipos de datos
 - Puede pasar funciones, almacenarlas en arreglos, etc.
 - Puede crear funciones anónimas
 - De vital importancia para Ajax
 - Las siguientes lineas son equivalentes:
 - `function foo(...) {...}`
 - `var foo = function(...) {...}`

Las funciones son tipos de datos

- Se pueden asignar funciones a variables
 - `function square(x) { return(x*x); }`
 - `var f = square;`
 - `f(5);` → 25
- Se pueden poner funciones en arreglos
 - `function doble(x) { return(x*2); }`
 - `var functs = [square, f , double];`
 - `functs[0](10);` → 100
- Se pueden pasar funciones a otras funciones
 - `algunaFuncion(square);`
- Se pueden regresar funciones de funciones
 - `function blah() { ... return(square); }`
- Se puede crear una función sin asignarla a una variable
 - `(function(x) {return(x+7);})(10);` → 17

Asignación de Funciones a Variables

- **Ejemplos**

```
function square(x) { return(x*x); }
```

```
var f = square;
```

```
square(5); → 25
```

```
f(5); → 25
```

- **Formas equivalentes**

```
function square(x) { return(x*x); }
```

```
var square = function(x) { return(x*x); };
```


Poniendo Funciones en Arreglos

- **Ejemplos**

```
var funcs = [square, f , double];
```

```
var f2 = funcs[0];
```

```
f2(7); → 49
```

```
funcs[2](7); → 14
```

- **Otras estructuras de datos**

- Las funciones también pueden ir en objetos o cualquier otra categoría de estructura de datos. No hemos visto objetos, pero aquí está un ejemplo rápido:

```
var objAleatorio = { a: 3, b: "Hi", c: square};
```

```
objAleatorio.a; → 3
```

```
objAleatorio.b; → "Hi"
```

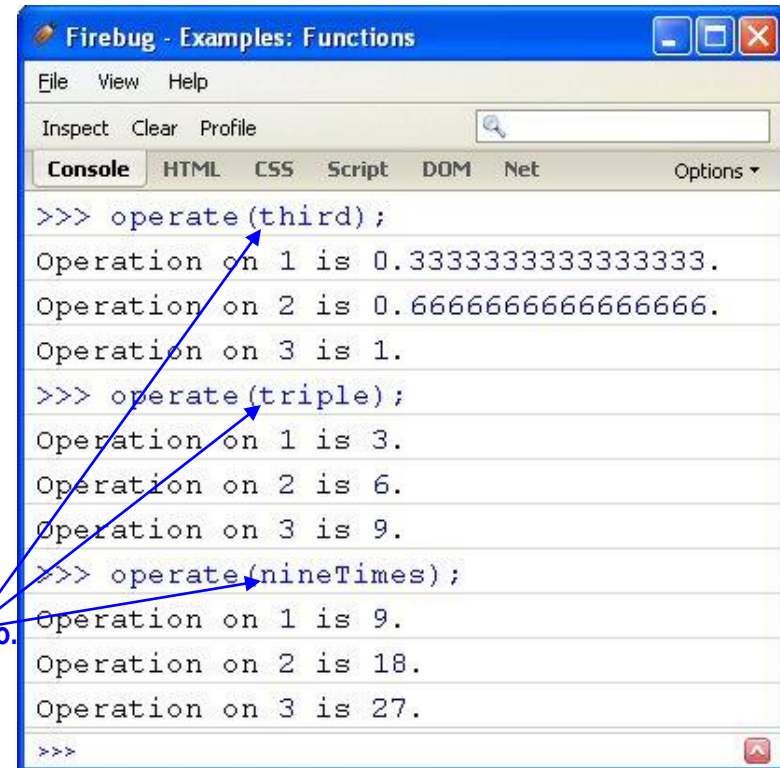
```
objAleatorio.c(6); → 36
```

Pasando Funciones a otras Funciones

```
function third(x) {  
    return(x / 3);  
}  
  
function triple(x) {  
    return(x * 3);  
}  
  
function nineTimes(x) {  
    return(x * 9);  
}
```

```
function operate(f) {  
    var nums = [1, 2, 3];  
    for(var i=0; i<nums.length; i++) {  
        var num = nums[i];  
        console.log("Operation on %o is %o.",  
                    num, f(num));  
    }  
}
```

Función como argumento.



Regresando Funciones de Funciones

- **Ejemplos**

```
function randomFunc() {  
  if(Math.random() > 0.5) {  
    return(square);  
  } else {  
    return(double)  
  }  
}  
  
var f3 = randomFunc();  
f3(5); // Regresa ya sea 25 o 10  
f3(5); // Regresa lo mismo que la linea anterior
```

- **Funciones creadas dinámicamente**

- En vez de una función predefinida como square, puede regresar una nueva función con `return(function(...) { ... });`

Se puede crear una Función sin asignarla a una Variable

- **Ejemplos**

`(function(x) {return(x+7);})(10);` → 17

```
function randomFunc2() {  
  if(Math.random() > 0.5) {  
    return(function(x) { return(x*x); });  
  } else {  
    return(function(x) { return(x*2); });  
  }  
}
```

- Mismo comportamiento que con la previamente mostrada `randomFunc`

- **Mas sobre funciones anónimas**

- Se denominan “cerraduras” si las funciones se refieren a variables locales de fuera. No se puede hacer Ajax sin ellas!



Funciones: Tópicos Avanzados

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Java 6, Ajax, jQuery, GWT, Spring, Hibernate, RESTful Web Services, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Funciones Anónimas con Datos Estáticos

- **Ejemplos**

```
function multiplicaPor7() {  
    return(function(n) { return(n*7); });  
}  
var f = multiplicaPor7();  
f(7); → 49
```

- **Forma equivalente**

```
function multiplicaPor7() {  
    var m = 7;  
    return(function(n) { return(n*m); });  
}  
var m = 700; // Valor de la variable global m es irrelevante  
var f = multiplicaPor7();  
f(7); → 49
```


Funciones Anónimas con Datos Capturados (Cerraduras)

```
function funcionMultiplicativa(m) {  
    return(function(n) { return(n*m); });  
}
```

```
var test = 10;  
var f = funcionMultiplicativa (test);  
f(7);    → 70  
test = 100;  
f(7);    → 70    // Aun regresa 70
```

Punto importante: cuando se llama a `funcionMultiplicativa`, se crea una función que tiene su propia copia *privada* de `m`. Esta idea de una función anónima que captura una variable local es la *única* forma de hacer Ajax sin tener problemas de variables globales.

Método apply : Uso Simple

- **Idea**

- Le permite aplicar funciones a arreglos de argumentos en vez de a argumentos individuales. Es un método *de* las funciones!
 - algunaFuncion.apply(null, arregloDeArgs);
- Mas adelante, veremos su uso con un objeto en vez de null

- **Ejemplos**

```
function hipotenusa(lado1, lado2) {  
    return(Math.sqrt(lado1*lado1 + lado2*lado2));  
}
```

```
hipotenusa(3, 4); → 5
```

```
var lados = [3, 4];
```

```
hipotenusa.apply(null, lados); → 5
```

```
Math.max.apply(null, [1, 3, 5, 7, 6, 4, 2]); → 7
```

Los métodos call y apply : Uso con Objetos

- **Idea**

- call

- Le permite llamar funciones sobre argumentos, estableciendo el valor de “this” primero.
 - Tendrá mayor sentido una vez que veamos objetos, pero la idea principal es que “this” le permite acceder propiedades de objetos. Por tanto, “call” trata a las funciones regulares como un método del objeto.

- apply

- Misma idea, pero proporciona argumentos como arreglo

- **Ejemplos**

```
function nombreCompleto() {  
    return(this.primerNombre + " " + this.apellido);  
}
```

nombreCompleto(); → "undefined undefined"

var persona = {primerNombre: "Bruce", apellido: "Wayne"};

nombreCompleto.call(persona); → "Bruce Wayne"



Fundamentos de Objetos

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Java 6, Ajax, jQuery, GWT, Spring, Hibernate, RESTful Web Services, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Fundamentos

- **Constructores**

- Funciones con el mismo nombre que la clase. Use “new”.
 - No hay una definición de clase separada! No hay POO “real” en JavaScript!
 - Puede definir propiedades con “this”
 - Debe usar “this” para las propiedades establecidas en los constructores
- ```
function MiClase(n1) { this.num = n1; }
var m = new MiClase(10);
```

- **Propiedades (variables de instancia)**

- No las define de manera separada
    - Cuando se refiera a una, JavaScript la crea
- ```
m.valor = 20; // Ahora m.num es 10 y m.valor es 20
```
- Usualmente mejor evitar introducir nuevas propiedades en código externo y mejor hacer la definición completa en el constructor

- **Metodos**

- Propiedades cuyos valores son funciones

Objetos: Ejemplo (Clase Circulo)

```
function Circulo(radio) {  
    this.radio = radio;  
  
    this.getArea =  
        function() {  
            return(Math.PI * this.radio * this.radio);  
        };  
}  
  
var c = new Circulo(10);  
c.getArea(); // Regresa 314.1592...
```


La Propiedad prototype

- **En el ejemplo previo**
 - Cada nuevo Circulo obtiene su propia copia de radio
 - Bien, ya que radio es un dato por Circulo
 - Cada nuevo Circulo tiene su propia copia de la función getArea
 - Malo, ya que la función nunca cambia y gastamos recursos
- **Propiedades a nivel clase**
 - `Nombreclase.prototype.nombrePropiedad = valor;`
- **Métodos**
 - `Nombreclase.prototype.nombreMetodo = function() {...};`
 - Caso especial de propiedades a nivel clase
 - Es legal hacer esto en cualquier parte, pero es mejor hacerlo en el constructor
- **Pseudo-Herencia**
 - La propiedad prototype puede ser usada para herencia
 - Complejo. Mas adelante lo veremos

Objetos: Ejemplo (Clase Circulo Actualizada)

```
function Circulo(radio) {  
    this.radio = radio;  
  
    Circulo.prototype.getArea =  
        function() {  
            return(Math.PI * this.radio * this.radio);  
        };  
}  
  
var c = new Circulo(10);  
c.getArea(); // Regresa 314.1592...
```



Métodos Static

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Java 6, Ajax, jQuery, GWT, Spring, Hibernate, RESTful Web Services, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Métodos Static (Espacios de Nombres)

- **Idea**

- Tener funciones relacionadas que no usen propiedades de objetos
- Desea agruparlas y llamarlas usando Utils.func1, Utils.func2, etc.
 - El agrupamiento es una conveniencia sintactica. No son métodos en realidad.
 - Ayuda a evitar conflictos de nombre cuando se mezclan librerías de JavaScript
- Similar a métodos static en Java

- **Sintaxis**

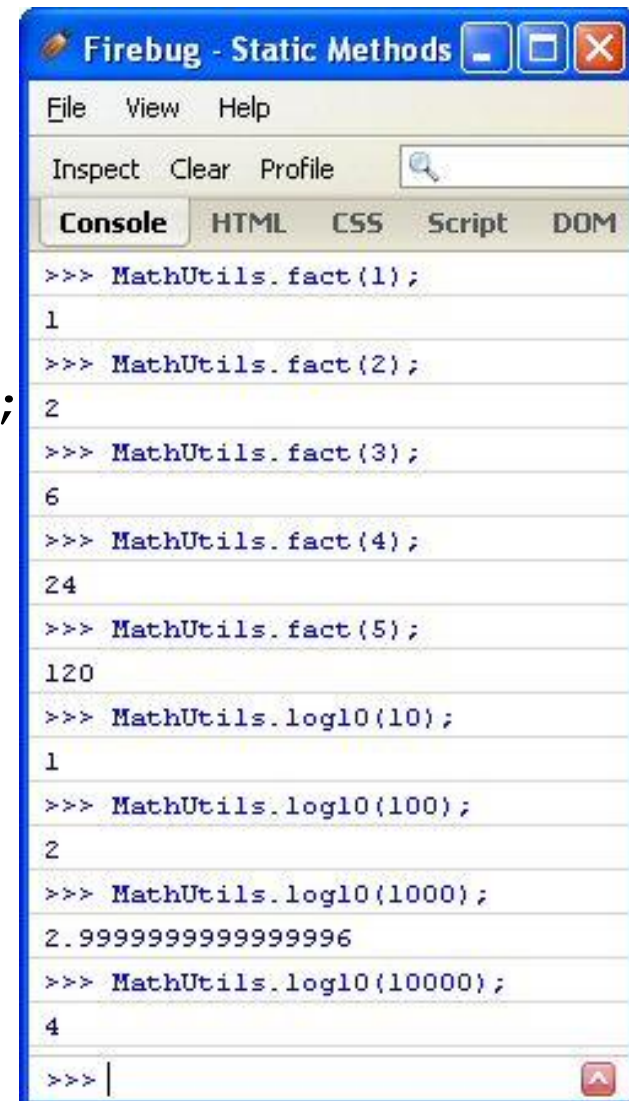
- Asigne funciones a propiedades de un objeto, pero no defina un constructor. Por ejemplo:
 - ```
var Utils = { }; // O new Object()
Utils.foo = function(a, b) { ... };
Utils.bar = function(c) { ... };
var x = Utils.foo(val1, val2);
var y = Utils.bar(val3);
```

# Métodos Static: Ejemplo(Código)

```
var MathUtils = {};
```

```
MathUtils.fact = function(n) {
 if (n <= 1) {
 return(1);
 } else {
 return(n * MathUtils.fact(n-1));
 }
};
```

```
MathUtils.log10 = function(x) {
 return(Math.log(x)/Math.log(10));
};
```



# Espacios de Nombres en Aplicaciones Reales

- **Las mejores practicas en proyectos grandes**
  - En muchos (si no es que la mayoría) proyectos grandes, *todas* las variables globales (incluyendo funciones!) están prohibidas debido a la posibilidad de colisiones de nombres con fragmentos hechos por diferentes autores.
  - Por tanto, estos espacios de nombres primitivos juegan el papel de los paquetes en Java. Mas debiles, pero de mucha ayuda.
- **Variacion: repetir el nombre**
  - `var MiApp = { };`
  - `MiApp.foo = function foo(...) { ... };`
  - `MiApp.bar = function bar(...) { ... };`
  - El nombre a la derecha no se hace global. La única ventaja es para la depuración
    - Firebug y otros ambientes mostrarán el nombre cuando imprima el objeto de función.





# JSON: Objetos Anónimos

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Servlets, JSP, JSF 2.0, Java 6, Ajax, jQuery, GWT, Spring, Hibernate, RESTful Web Services, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# JSON (JavaScript Object Notation)

- **Idea**

- Es una representación textual simple de los objetos de
  - Denominados “literales de objetos” u “objetos anónimos”
- Principales Aplicaciones
  - Objetos usados una sola vez (en vez de clases reusables)
  - Objetos recibidos via strings

- **Directamente en JavaScript**

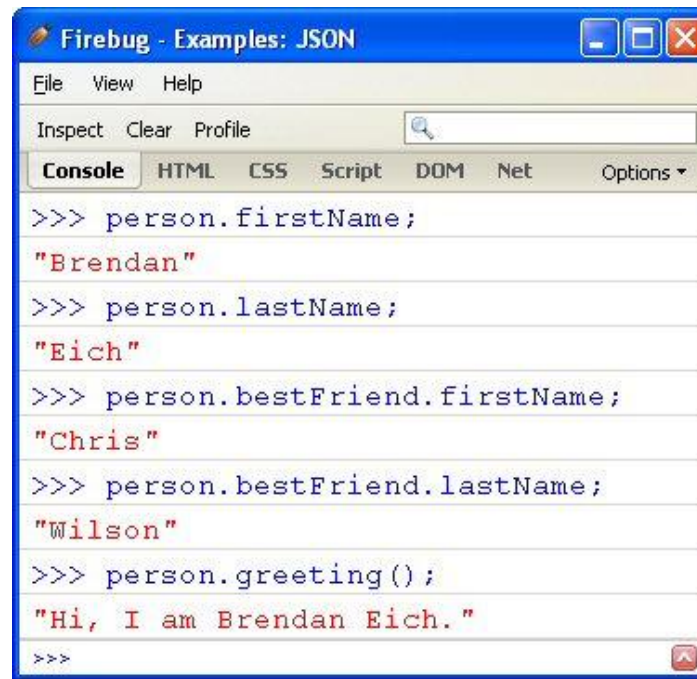
- ```
var someObject =  
  { property1: value1,  
    property2: value2,  
    ... };
```

- **En un string (por ejemplo, recibidos de la red)**

- Rodee la representación del objeto en parentesis
 - Pase tal representación a la función “eval”

JSON: Ejemplo

```
var person =  
  { firstName: 'Brendan',  
    lastName: 'Eich',  
    bestFriend: { firstName: 'Chris',  
                  lastName: 'Wilson' },  
    greeting: function() {  
      return("Hi, I am " + this.firstName +  
             " " + this.lastName + ".");  
    }  
  };  
};
```



JSON Estricto

- **JSON en la práctica**
 - Es la forma en que escribiría una estructura de datos en JavaScript.
 - Es lo que soportan “eval”, Prototype, y jQuery 1.3
- **JSON estricto de acuerdo a json.org**
 - Subconjunto de JavaScript donde:
 - Los nombres de propiedades del objeto deben estar entre comillas dobles
 - Los strings son representados con comillas dobles solamente (no apostrofes)
 - Es lo que soporta jQuery 1.4. Ya que esto es lo que claramente se describe en json.org, debería seguir este formato cuando envíe JSON desde el servidor.
- **Tipo MIME para JSON desde el servidor**
 - El RFC 4627 dice que JSON debe tener el tipo "application/json"
 - No hay librerías conocidas que hagan obligatorio esto

Internet Explorer y las Comas Extra

- **Firefox & Chrome toleran comas al final**
 - Toleradas tanto en arreglos como en objetos anónimos
 - `var nums = [1, 2, 3,];`
 - `var obj = { primerNombre: "Jose", apellido: "Flores", };`
- **IE fallará en ambos casos**
 - Por portabilidad, debería escribir *sin* comas después del elemento final :
 - `var nums = [1, 2, 3];`
 - `var obj = { primerNombre: "Jose", apellido: "Flores"};`
 - Este problema aparece a menudo, especialmente cuando se construyen datos JSON en el servidor.

Otros Trucos de Objetos

- **Operador instanceof**

- Determina si un elemento es miembro de una clase
 - ```
if (blah instanceof Array) {
 hazAlgoCon(blah.length);
}
```

- **Operador typeof**

- Regresa el tipo directo del operando, como un String
  - "number", "string", "boolean", "object", "function", or "undefined".
    - Los arreglos y null ambos regresan "object"

- **Agregar métodos a clases existentes**

```
String.prototype.describeLongitud =
 function() { return("Mi longitud es " + this.length); };
"Cualquier String aleatorio".describeLongitud();
```

- **eval**

- Toma una representación String de *cualquier sentencia* JavaScript y la ejecuta
  - ```
eval("3 * 4 + Math.PI"); // Regresa 15.141592
```


Mas sobre eval

- **Strings simples**

- Simplemente se pasan a eval
- `var test = "[1, 2, 3, 2, 1].sort()";`
- `eval(test);` → `[1, 1, 2, 2, 3]`

- **Strings delimitados con { ... }**

- Debe agregar paréntesis extras para que JavaScript sepa que las llaves son de literales de objetos, y no para delimitar sentencias.
- `var test2 = "{ primerNombre: 'Juan', apellido: 'Soto' }";`
- `var persona = eval("(" + test2 + ")");`
- `persona.primerNombre;` → `"Juan"`
- `persona.apellido;` → `"Soto"`



Funciones con un Numero Variable de Argumentos

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Java 6, Ajax, jQuery, GWT, Spring, Hibernate, RESTful Web Services, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Variable Args: Resumen

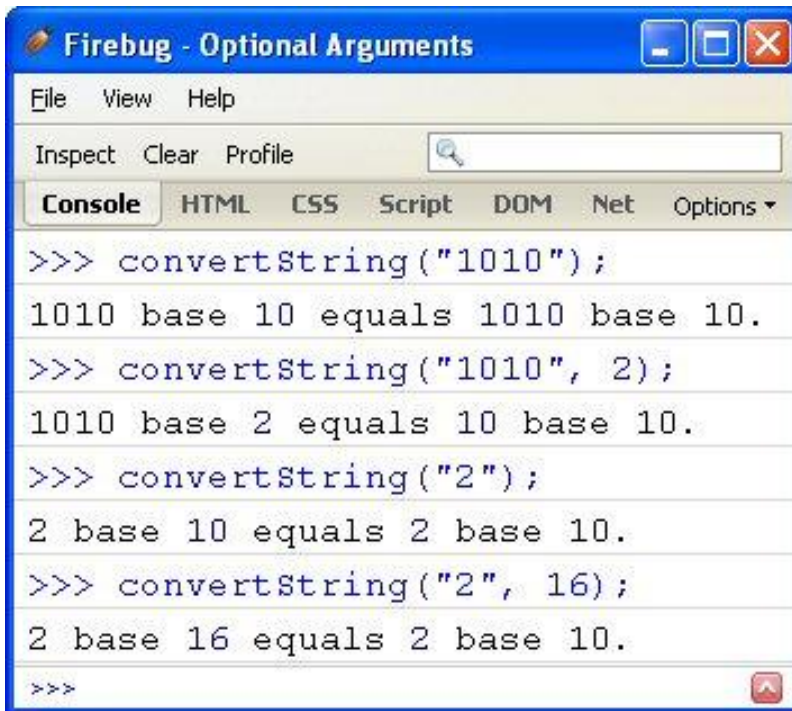
- **Número fijo de args opcionales**
 - Las funciones pueden *siempre* ser llamadas con cualquier número de args
 - Compare el tipo de args con "undefined"
 - Como ejemplo esta la funcion `convertString` más adelante
- **Args arbitrarios**
 - Descubre el número de args con `arguments.length`
 - Obten los argumentos via `arguments[i]`
 - Como ejemplo esta la función `longestString` más adelante
- **Args opcionales via objetos anónimos**
 - El llamado debe siempre proporciona el mismo número de argumentos, pero uno de los argumentos es un objeto anónimo (JSON)
 - Este objeto tiene campos opcionales
 - Este es el enfoque mayormente usado para librerías de usuario
 - Como ejemplo, esta la función `sumNumbers` más adelante

Args Opcionales: Detalles

- **Puede llamar a *cualquier* función con cualquier número de argumentos**
 - Si se llama con menos args, los args extra estan indefinidos (undefined)
 - Puede usar `typeof arg == "undefined"` para esto
 - También puede usar una comparación booleana si está seguro que ningún valor real pudiera coincidir (por ejemplo, 0 y undefined ambos regrean true para `!arg`)
 - Use comentarios para indicar que hay args opcionales a los desarrolladores
 - `function foo(arg1, arg2, /* Opcional */ arg3) {...}`
 - Si se llama con args extra, puede usar el arreglo “arguments”
 - Independientemente de las variables definidas, `arguments.length` le dice cuantos argumentos fueron proporcionados, y `arguments[i]` regresa el argumento i.
 - Use comentarios para indicar args variables (varargs)
 - `function bar(arg1, arg2 /* varargs */) { ... }`

Argumentos Opcionales

```
function convertString(numString, /* Optional */ base) {  
    if (typeof base == "undefined") {  
        base = 10;  
    }  
    var num = parseInt(numString, base);  
    console.log("%s base %o equals %o base 10.",  
                numString, base, num);  
}
```



Varargs

```
function longestString(/* varargs */) {  
    var longest = "";  
    for(var i=0; i<arguments.length; i++) {  
        var candidateString = arguments[i];  
        if (candidateString.length > longest.length) {  
            longest = candidateString;  
        }  
    }  
    return(longest);  
}
```

`longestString("a", "bb", "ccc", "dddd");` → "dddd"

Usando JSON para los Argumentos Opcionales

- **Idea**

- El llamador siempre proporciona el mismo número de argumentos, pero uno de ellos es un objeto anónimo (JSON)
 - Este objeto tiene campos opcionales
- Este enfoque es ampliamente usado en Prototype, Scriptaculous, y otras librerías de JavaScript

- **Ejemplo (a/b: requeridos, c/d/e/f: opcionales)**

- `algunaFuncion(1.2, 3.4, {c: 4.5, f: 6.7});`
- `algunaFuncion(1.2, 3.4, {c: 4.5, d: 6.7, e: 7.8});`
- `algunaFuncion(1.2, 3.4, {c: 9.9, d: 4.5, e: 6.7, f: 7.8});`
- `algunaFuncion(1.2, 3.4);`

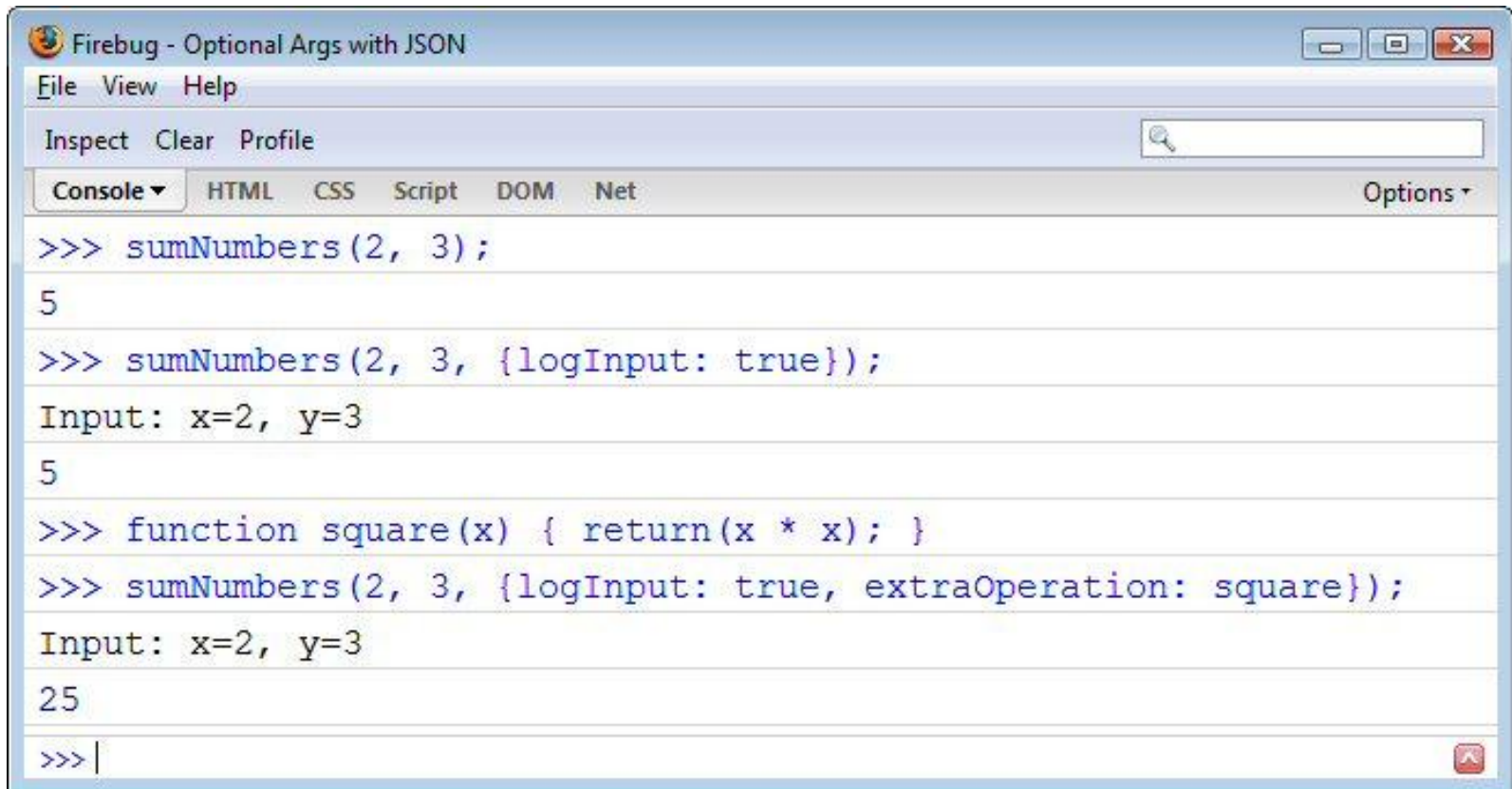
Usando JSON para los Argumentos Opcionales: Ejemplo

```
function sumNumbers(x, y, extraParams) {  
  var result = x + y;  
  if (isDefined(extraParams)) {  
    if (isTrue(extraParams.logInput)) {  
      console.log("Input: x=%s, y=%s", x, y);  
    }  
    if (isDefined(extraParams.extraOperation)) {  
      result = extraParams.extraOperation(result);  
    }  
  }  
  return(result)  
}
```

```
function isDefined(value) {  
  return(typeof value !== "undefined");  
}
```

```
function isTrue(value) {  
  return(isDefined(value) && (value == true))  
}
```

Usando JSON para los Argumentos Opcionales: Resultado del Ejemplo



The screenshot shows the Firebug console window titled "Firebug - Optional Args with JSON". The console is set to the "Console" tab. The following code is executed:

```
>>> sumNumbers(2, 3);  
5  
>>> sumNumbers(2, 3, {logInput: true});  
Input: x=2, y=3  
5  
>>> function square(x) { return(x * x); }  
>>> sumNumbers(2, 3, {logInput: true, extraOperation: square});  
Input: x=2, y=3  
25  
>>> |
```

The console output shows the results of these calls: the first call returns 5; the second call returns "Input: x=2, y=3" followed by 5; the third call returns "Input: x=2, y=3" followed by 25.

Resumen

- **General**

- No trate universalmente de usar el estilo Java cuando programe en JavaScript. Si lo hace, no aprovechará al máximo las características de JavaScript.

- **Funciones**

- Totalmente diferentes de Java. Pasar funciones a funciones y hacer funciones anónimas es muy importante.
 - No lo considere raro o inusual, sino como una práctica normal.

- **Objetos**

- Constructor define la clase. Use “this”. Use prototype para los métodos.
 - Totalmente diferente a Java. No es como la POO clásica.

- **Otros trucos**

- `algunaFuncion.apply(null, arregloDeArgs);`
- `var algunValor = eval("(" + algunString + ")");`
- Varias formas de hacer args opcionales. Literales de objetos a menudo la mejor opción.



Preguntas?

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Java 6, Ajax, jQuery, GWT, Spring, Hibernate, RESTful Web Services, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.