Web

Master 1 IFI

Andrea G. B. Tettamanzi

Université de Nice Sophia Antipolis

Département Informatique

andrea.tettamanzi@unice.fr

Unit 4

Client-Side Programming (JavaScript and the HTML5 API)

Agenda

- Elements of the JavaScript language
- JavaScript and HTML5
- JavaScript's Object-Oriented Model
- Integrated Objects
- Error Handling

Historical Remarks

- Created in 1995 by Brendan Eich for Netscape
- In December 1995, Sun and Netscape announce its release
- In March 1996, JavaScript engine in NS Navigator 2.0
- Microsoft strikes back by developing Jscript, in August in IE
- In 1997, JavaScript becomes an ECMA standard: ECMAScript
- Choice of the name
  - Marketing reasons
  - Complementary to Java
  - Confusion in the general public

Language Overview

- Object-oriented, but based on prototypes. No classes
- A program is a collection of communicating objects
- Object = collection of properties
- Primitive values: undefined | null | Boolean | Number | String
- Collection of integrated objects:
  - Global Object
  - Object, Function, Array, String, Boolean, Number, Math, Date, RegExp, JSON
  - Several error objects: Error, EvalError, …
- Operators: arithmetic, relational, logical, etc.
- Syntax purposefully similar to Java, but less strict

Types and Expressions

- JavaScript uses weak and dynamic typing
- The type of an expression is determined by its result
  - Numbers: integers or floating-point
  - Character strings
  - Boolean (false, true)
  - « null » a single-valued type = absence of data
- Simple expressions: formed by a single element
  - Literal
  - Variable identifier
  - The keyword “this”
- Complex expressions: built w/ operators / functions / methods

Statements

The ECMAScript standard provides for 15 distinct statement types:

- Block
- VariableStatement
- EmptyStatement
- ExpressionStatement
- IfStatement
- IterationStatement
- ContinueStatement
- BreakStatement
- ReturnStatement
- WithStatement
- LabelledStatement
- SwitchStatement
- ThrowStatement
- TryStatement
- DebuggerStatement

Control Structures

- The same as Java – that's on purpose!

- Conditional
  - if(*cond*) … [else …]
  - switch(*expr*) { case … : … break ; … default : … }
- Iterative
  - while(cond) …
  - do ... while(*cond*) ;
  - for(*instr* ; *cond* ; *instr*) …

Function Definition

- Anonymous function definition (it's a "lambda": λx.y)
  function(*arguments*) { … }

- Named function definition:
  function *name*(*arguments*) { … }

- Returning a result:
  return(*expression*);

- Function call:
  name(*arguments*)

JavaScript and HTML

- JavaScript interacts with its containing HTML page via:
  - The <script> tag
  - The dialog-box functions
  - The events
  - The document object model (DOM)
  - The HTML5 API
    - Local Storage
    - Geo-location
    - Drag and Drop
    - Web Sockets

The <script> Tag

- JavaScript code is attached to an HTML document by the <script> tag, which may be placed anywhere
- Two ways to do that (in HTML5) :
  - Embedded script (i.e., the code is contained within the HTML document),
    <script> *embedded code* </script>

    Example:
    <script>document.write("Hello World!")</script>
  - External script (i.e., the code is in a separate file),
    <script src="*URL of the script file*"> </script>
- In HTML 4, type was mandatory: <script type="text/javascript">

Dialog Boxes

- JavaScript can pop up three types of dialog boxes
- Alert dialog boxes
  - **alert**(*message*)
  - Displays a dialog box with a message and an [OK] button
- Confirmation dialog box
  - **confirm**(*message*) → true | false
  - Displays a dialog box with a message and [Cancel] [OK]
- Prompt dialog box
  - **prompt**(*message*, *default_text*) → user input
  - Displays a message, a text box, which the user can type into, initialized with the provided default text, and the [Cancel] and [OK] buttons

HTML Events

- HTML events are user actions that can cause an interaction
- Examples: mouse click, mouse move, etc.
- JavaScript code can be associated to events
- Such association is made by means of *event handlers*

- Example:

```
<button type="button" onclick="calc_input(1)">1</button>
```

- Every HTML element supports a given set of events

Objects

- A JavaScript object can be thought of as a set of (property, value) pairs, which can be represented with the intuitive notation

  property : value
- This notation is also used by the syntax of the language to define object literals
- Properties can be
  - Own: explicitly defined in an object
  - Inherited: not explicitly defined in an object, but implicitly derived from other objects (called *prototypes*)
- A property may take values of any type
- JavaScript objects are implemented as hash tables

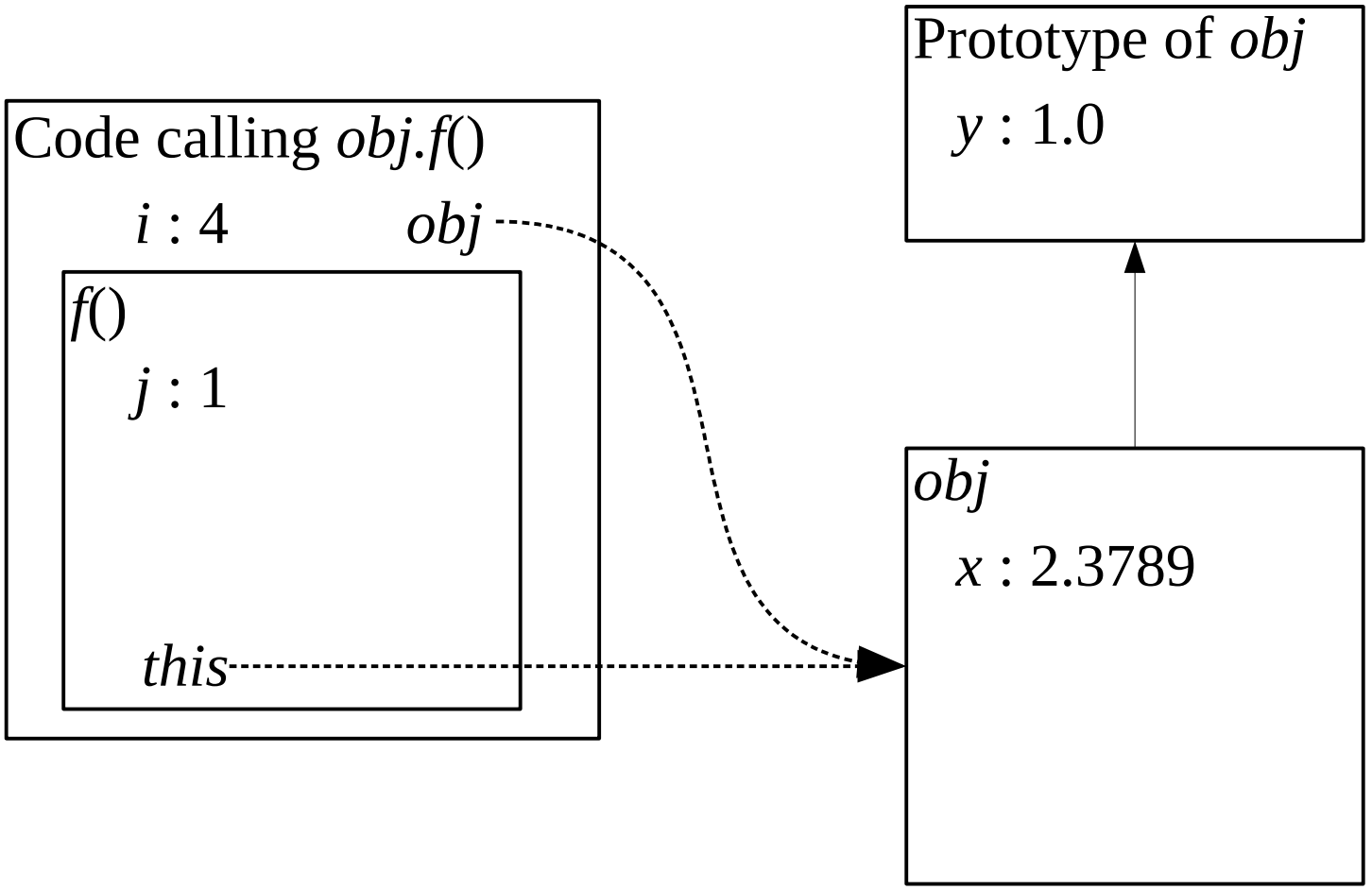
Methods

- A property of an object can have a function as its value
- One calls it a “method” in that case
- When a method of an object is called, the special variable “this” becomes a reference to the object
- Because of that, the code within the method can access the properties of the object via the “this” variable

Lexical Scoping

- When a function is called, its code is executed in a “lexical environment”
- If the function is called as a method of an object, the “this” variable becomes a reference to that object; otherwise, it references the global object
- The identifiers defined in the outer lexical environments (for instance, the one of the code calling the function) are still within the scope

Lexical Scoping



Encapsulation

- Fundamental Principle of OOP
  - Hide the details about data representation
  - Allow access to data only through methods
  - ⇒ The interface never exposes the data, only methods
- Certain OOP languages (e.g., C++, Java) provide access modifiers for the methods and data of a class:
  - Public – visible from any class, everywhere
  - Protected – visible from the class or a subclass only
  - Private – visible from the same class only

Encapsulation in JavaScript

- In JavaScript, the properties (thus including methods) of an object are all “public”, as it were
  - Any piece of code may access and/or change them
- This does not exclude to apply the encapsulation principle
  - As a programming good practice
  - As a form of self-discipline
  - Workarounds are always possible, but should be avoided
- Such « primitive » form of encapsulation is based on
  - Getters: methods that read and return the value of an attibute
  - Setters: methods that change (set) the value of an attribute
- These methods can check the legality or compute the values

Private Properties and Methods

- Ordinary variables of the constructor may be used as private members of the constructed objects
  - These variables remain attached to the objects
  - They are not visible outside of it
  - This holds for parameters passed to the constructor as well
- In addition, they are not visible to the ordinary methods of the constructed objects either
- However, they can be accessed by the constructor's internal functions
  - These function are also invisible outside
  - We might call them "private methods"

Example

```
function Container(param) {
     function dec() {
          if (secret > 0) {
               secret -= 1;
               return true; }
          else return false; }
     this.a = param;
     var secret = 3;
     var that = this;
}
```

Privileged Methods

- A privileged method
  - Has the right to access private properties and methods
  - Can be called by ordinary methods
  - Can be called by external code
- One can delete or replace a privileged method
  - However, nobody can change it
  - Nor force it to reveal its secrets
- Privileged method definition: in the constructor,
  - Assign a constructor's internal function to a public property (in the lexical scope of **this**)

Example

```
function Container(param) {
  function dec() {
    if (secret > 0) {
      secret -= 1;
      return true; }
    else return false; }
  this.service = function () {
    return dec() ? that.a : null; };
  this.a = param;
  var secret = 3;
  var that = this;
}
```

Prototypes and Heritage

- Every object has an internal link towards its prototype
- A prototype is an object and can in turn have a prototype
- We are then talking about the prototype chain of an object
- Given object o, the value o.x of property x is determined as follows:
  - if x is an own property of o, o.x returns the value of property x in o;
  - else, the prototype chain of o is followed up to the first prototype p where x is defined; in this case, o.x returns the value of p.x;
  - if, while following the prototype chain, the “null” object is reached, o.x returns “undefined”.

Creating Objects

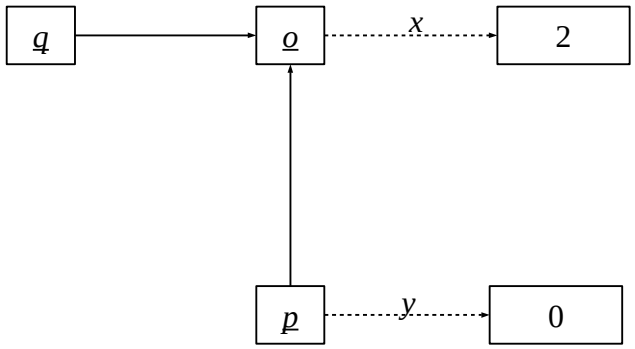
- An object can be created in various ways, including
  - Syntactically, by means of the literal notation:

    obj = { x : 5, y : 10 } ;
  - By a constructor
    - A constructor, in JavaScript, is a function that is called with the “new” operator:

      new f(…) ;
    - It returns a new object having f.prototype as its prototype
  - By calling a utility method of Object:
    - Object.create(*proto*).

Example

```
o = { x : 2 };
function f() { … }
f.prototype = o;
p = new f();
p.y = 0;
q = new f();
```



Integrated Objects

- A number of integrated objects are available when a JavaScript program is executed.
- The global object is directly belongs to the program's lexical environment
  - It contains the "global" variables and functions
- The other integrated objects can be accessed as initial properties of the global object
- Many integrated objects are functions
  - They can be called with arguments
  - Some can be used as constructors
- The names of these objects are heavily inspired by the names of the classes of the Java platform and of their methods and attributes

The "Object" Object

- Allows to create a conversion object for a given value
- By default, the prototype of the created objects is "null"
- Its methods include
  - create(proto, o): creates a new object with the specified prototype and all the properties of object o
  - keys(o): returns an array containing the names of all the enumerable properties of the given object
  - getPrototypeOf(o): returns the prototype of object o
  - setPrototypeOf(o, p): sets the prototype of object o to p
  - assign(destObj, srcObj): copies all the enumerable properties for the source object to the destination object

The "Function" Object

- Any function in JavaScript is in fact an object inheriting the prototype of Function
- As a constructor, Function creates functions at run-time
- The methods of the objects constructed by Function include:
  - apply(o [, args]): applies the function as if it were a method of object o; the arguments of the function have to be passed as an Array object
  - bind(o): creates a new function which, when called, invokes this function as if it were a method of object o
  - call: the same as apply, but with arguments passed one by one instead of as one Array object

Error Handling

- The error handling mechanism relies on three ingredients:
  - One ore more exception handlers;
  - A mechanism to raise (= signal) exceptions;
  - A mechanism allowing to associate exceptions to their handlers.
- In JavaScript, the mechanism to associate exceptions to their handlers is provided the “try … catch … finally” structure
- The exception handlers are contained in the “catch” clause of this structure
- Exceptions are raised by means of the “throw” operator and objects created by the Error constructor.