

Introduction to JavaScript For Modern Software Development

Kenneth Geisshirt
<http://kenneth.geisshirt.dk/>

Agenda – before lunch

- History and standard
- Implementations
- Tools
- Syntax
- Procedural programming
- Object-orientation
- Functions as first-class object
- Fluent interfaces

- Syntax highlighting in most editors
- JSLint is a great code checker
 - <http://jshint.com/>
 - Easy to run using Rhino and integrate with Emacs
- Firebug has a nice Javascript debugger
 - <http://getfirebug.com/>
- Web developers: use a framework:
 - JQuery - <http://jquery.com/>
 - YUI - <http://developer.yahoo.com/yui/>
 - Prototype - <http://prototypejs.org/>
 - Dojo - <http://www.dojotoolkit.org/>
 - Underscore - <http://documentcloud.github.com/underscore/>

Agenda – after lunch

- JavaScript and the browser
 - Interaction with DOM
 - JSON
 - AJAX
- Moderne web programming
 - JQuery
- Back-end programming
 - Node.js

Features

- Dynamic
 - Extending and modifying a running program
- Weakly typed
 - Implicit type conversion
 - Implicit declared variables
- Prototype-based objects
 - Objects but no classes
- Functions as first-class objects
 - A Function is just an object

History and standard

- Designed by Brendan Eich (Netscape) in 1995
 - Netscape Communicator 2.0b3 (December 1995)
 - Internet Explorer 3.0 (August 1996)
- JavaScript is a Oracle trademark
- Standardized by ECMA
 - Official name ECMAScript
 - ECMA-262/ISO-16262 (1997)
 - Revision 5.1 is latest revision (June 2011)

Implementations

- Closed source
 - JScript (Microsoft)
 - Futhark (Opera)
- Open Source
 - SpiderMonkey (Mozilla, written in C)
 - Rhino (Mozilla, written in Java)
 - QtScript (Nokia Troll Tech)
 - JavaScriptCore (Apple)
 - V8 (Google)

Future implementations

- *Just-In-Time* (JIT) is becoming popular
 - V8 is doing it already
 - TraceMonkey and JägerMonkey (Mozilla)
 - SquirrelFish (Apple)
 - Carakan (Opera)
 - Tamparin (Adobe and Mozilla)
- Size does matter (smaller is better)
 - Transformation and compression of code
 - Google Closure

Basic syntax

Syntax – comments and lines

- Just like C
 - The rest of the line: `//`
 - Multi-line: `/* */`
- `//` is recommended by Javascript developers
- Statement separator is `;` (semicolon)
- Statements can span multiple lines
- White spaces: space and tab
- Case sensitive

Syntax – literals

- Booleans
 - Examples: `true`, `false`
 - Other false values: `null`, `NaN`, `' '`, `0`, `undefined`
- Numbers
 - Integers
 - Examples: `42`, `101010`
 - Real numbers
 - Examples: `3.1415926535`, `6.022e23`, `6.626e-34`
- Strings
 - Examples: `"Hello"`, `'world'`, `"Don't panic"`

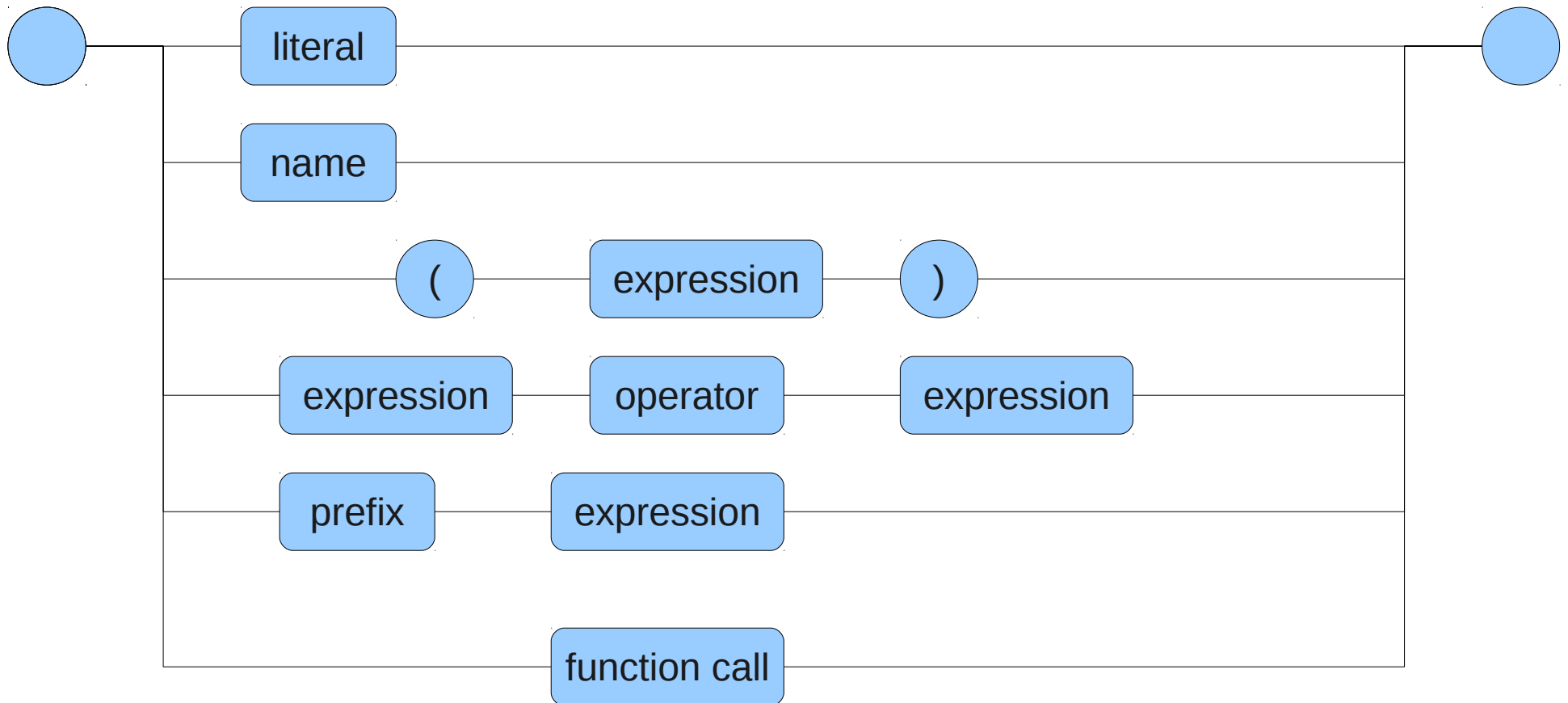
Syntax - names

- Begin with a letter
 - can be followed by letters, digits and underscores (`_`)
 - Many implementations allow dollar and underscore as first character

Reserved words

abstract, boolean, break, byte, case, catch, char, const, continue, debugger, default, delete, do, double, else, enum, export, extends, false, final, float, for, function, goto, if, implements, import, in, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, typeof, var, volatile, void, while, with

Syntax - expression



"hello "+"world", f(42), 1.0/(x+y), -3.14, 5>7

Example: expr.js

· osd12-js.odp

Syntax – expression

- `typeof(·)` returns type of the argument as string
 - `undefined`, `boolean`, `number`, `string`, `object`, `function`
- Equality: `===` and `!==`
- Inequality: `<`, `>`, `>=`, `<=`
- Logical: `&&` (and), `||` (or), `!` (not)
- Arithmetic: `+`, `-`, `*`, `/`, `%`
- String concatenation: `+`
- `+=` and `-=` can be used (but not recommended)

Example: `opers.js`

== VS ===

- == compares values
- === compares values **and** types
- Similar for != and !==
- Recommendation: use === and !==

Example: equal.js

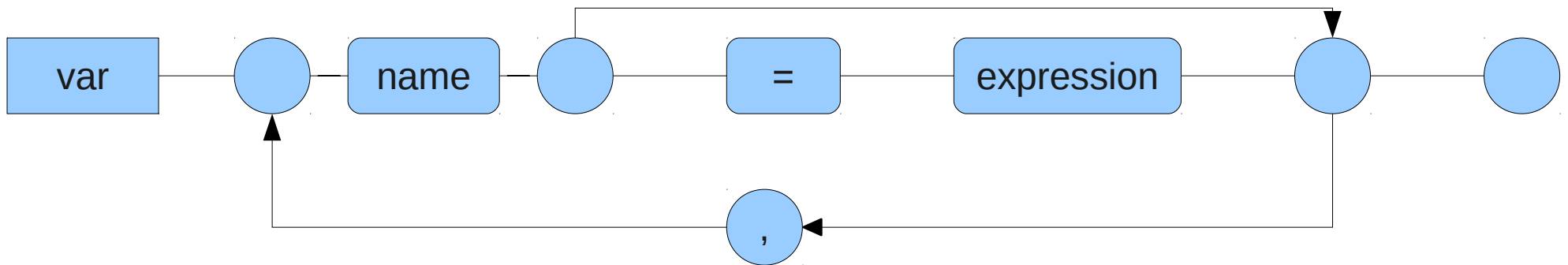
Arrays

- Two ways to create an array
 - Recommended: `a = [2.7182, 3.1415]`
 - `a = new Array()`
- Zero based index
- Elements do not have to be of the same type
- Indirection by `[]`
- Property `length` is the number of elements

Example: `array.js`

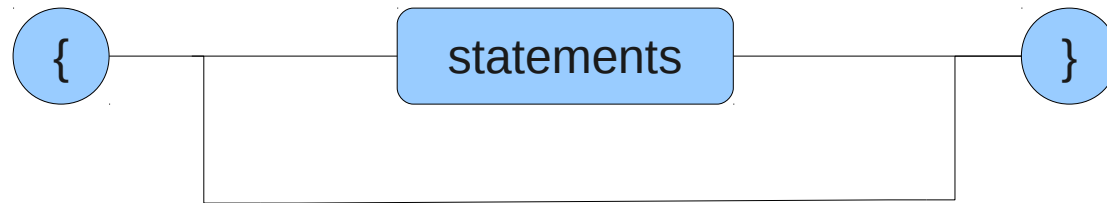
Procedural programming

Procedural programming



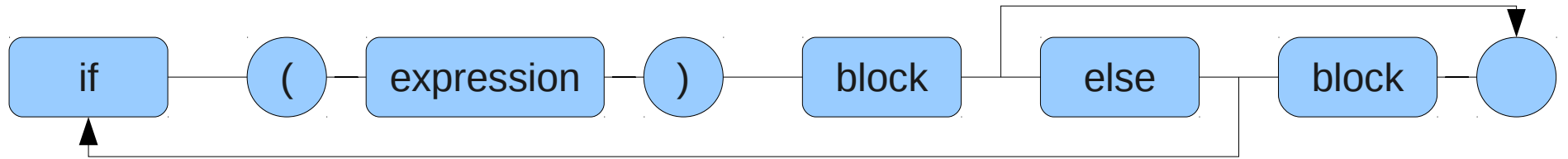
- Declare variables
 - Only global name space
 - No late declarations
 - Visible in all function
 - Interpreter deduces type at runtime

Blocks



- Block is similar to blocks in C
 - But they do not declare a scope
 - Variables are visible **before** and **after** the block
 - Advice: declare all variables in the top of functions

Branches



- Falsy values for an expression:
 - false, null, undefined, "", 0, NaN
 - All other values are true

Example: if.js

Other branches

- The switch construct is similar to C
 - Case expressions can be variable
 - And can be a number or a string
 - Each case must be terminated by break

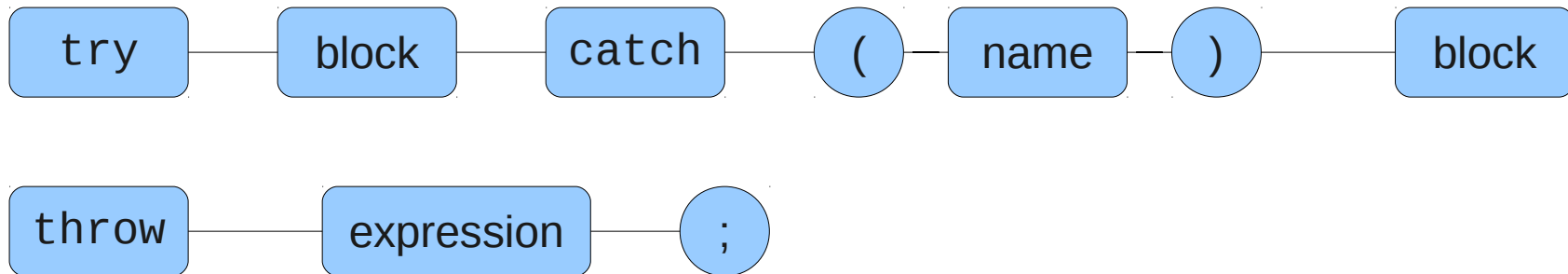
```
switch (c) {  
  case 'A':  
    break;  
}
```

Example: switch.js

- Four loops constructions exist
 - Execute `while` an expression is true
 - Zero or more times
 - `do while` an expression is true
 - At least one time
 - A for loop while in C
 - A for each (`for ... in`) style of loop
 - Order is not guaranteed

Example: loops.js

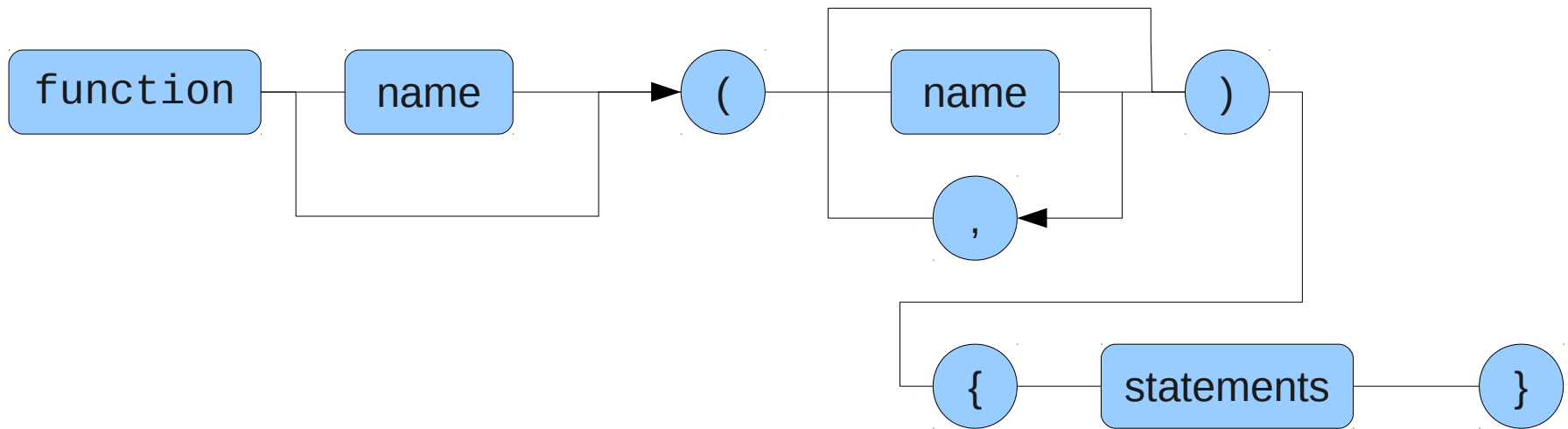
Exceptions



- Execution of a block (`try`)
 - All exceptions handled (`catch`)
 - by an exception object (`name`)
- Any object can be used as exception (`throw`)

Example: `exception.js`

Functions



- Recursion is allowed
- Functions can be embedded within function (local function)
- Any object can be returned

Objects

Objects

- Basic object concepts
 - Javascript has objects
 - Javascript has no classes
- Mother of all objects: `Object`
- Everything is an object
 - Booleans, numbers, strings, arrays, functions
- An object is a set of properties or key/value pairs
- Values can be all kind of objects

Objects

- Create an object: `new Object()`
- Indirection: `.` (dot) or `[]`
- Remove a property using `delete`
- No protection of properties

Methods

- Values can be (anonymous) functions
 - We call them methods
- The keyword `this` refers to the object in methods
- Methods can have internal (aux) functions

Example: `objects.js`

Simple constructor

- Define a function

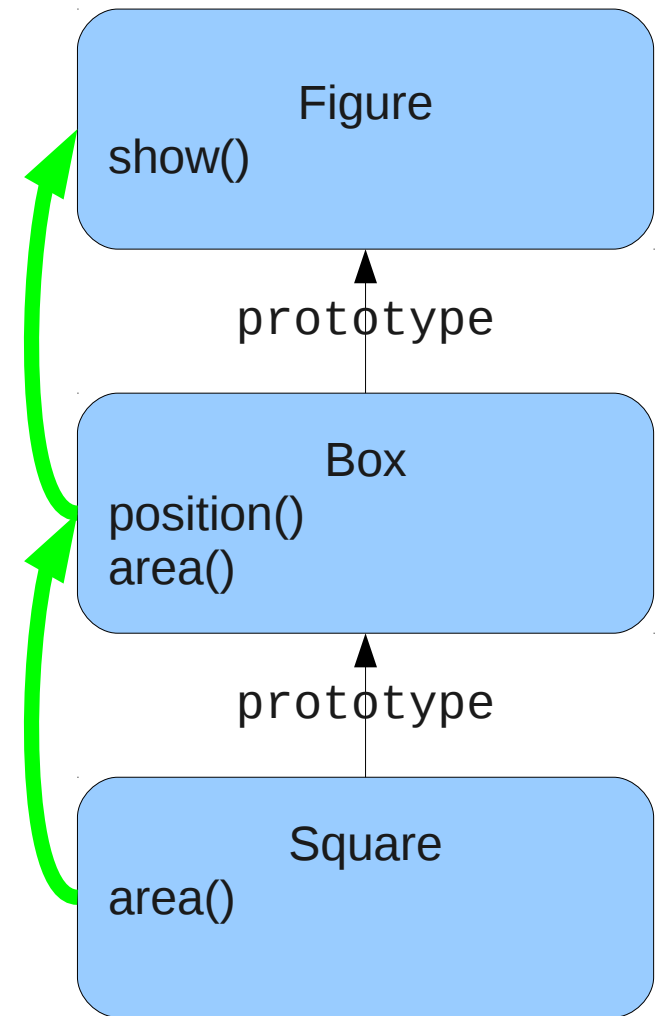
```
function Graph(title) {  
    this.title = title;  
}
```

```
var g = new Graph('TT');
```

- Use capital letters for such objects!

Prototype

- Every object has a link to a parent
 - The property is named `prototype`
- Differential inheritance
 - New object is specified by the difference from its prototype/parent
 - A long chain of prototypes (delegation)



Square.show()

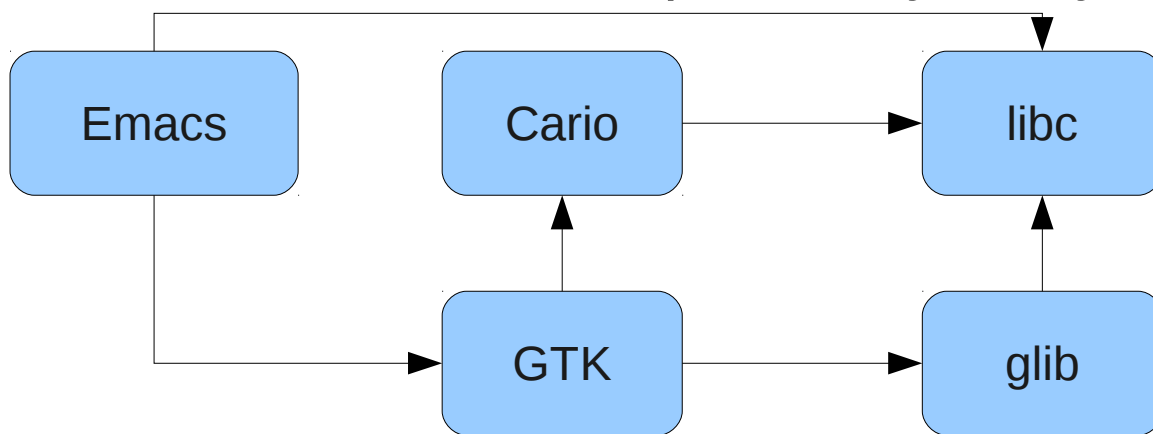
Reflection and Enumeration

- `typeof` returns the type of an object (as string)
- The method `hasOwnProperty` returns true if object (and not parent) has property
- Operator `in` can give you all properties

```
for (name in obj) {  
    if (typeof(obj[name]) !== 'function') {  
        console.log(name+' : '+obj[name]);  
    }  
}
```

Example: graphs

- A graph is
 - A set of nodes and a set of edges (between nodes)
- Directed Acyclic Graph (DAG)
 - Edges have a direction
 - And no loops
 - Often used for dependency analysis



Example: graph.js
· osd12-js.odp


Fluent interfaces

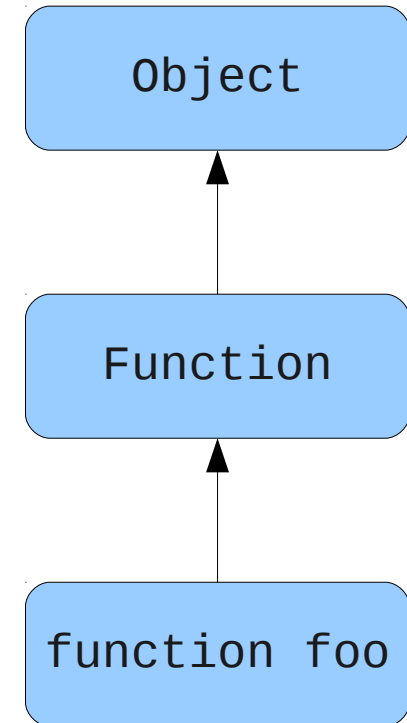
- Common in OOP and JavaScript
 - Coined by Martin Fowler
 - Chaining method invocations
 - ⇒ more readable code
 - Every method returns reference to object

Example: fluent.js

Functions

Functions

- Functions are objects 
 - Functions can be arguments to functions
 - And return value
- Scope is a bit tricky
 - Function scope and *not* block scope
 - Inner functions can access outer function's variables



Anonymous

- Function without a name
- Assign as object to variable
- Useful for callbacks

Example: `anonymous.js`

Arguments

- Listed and named parameters are most common
- `arguments` stores all parameters
 - Variable number of arguments
 - Order of parameters and array matches

The meaning of `this`

- Implicit parameter `this` refers to the current context
- Four different invocation patterns
 - Method
 - Function
 - Constructor
 - Apply
- Current context depends on invocation pattern

Invocation - Method

- Property in an object
 - Key concept in OOP
- . (dot) is used to refer to function
- `this` is set to the object
 - Binding of `this` is done at invocation time
 - Access to properties added *after* adding method

Invocation - Function

- Classical procedural programming
 - This is set to global scope
 - Even for inner functions
- Work-around is to save a reference to `this`
 - First statements of outer function:




```
var that;  
that = this;
```


Invocation - Constructor

- Functions can be used as constructors
 - The new operator create an object
 - Function's prototype is used
 - `this` is set to the object
 - `return` will return the object (`this`)
- Close to class-based OOP

Invocation - Apply

- The `Function` object has an `apply` method
 - Yes, functions have methods 
- `apply` has two parameters
 - Context – or just simply `this`
 - Arguments – stored in `arguments`

Closure

- Functions have access to the context when created
- Context will live as long as the function
- Useful patterns
 - Return a function
 - Callbacks
 - Anonymous function as parameter to function
 - Has access to context

Functional programming

- Currying and closures
 - Rewrite functions: $g_x(y) = f(x, y)$
 - Compose functions: $h = f \circ g$
 - Extend `Function` with a `curry` method

Examples: `highfunc.js`

JavaScript and the browser

Debugging

- Firebug is an excellent tool
 - Debugger (break point, single stepping)
 - HTML and CSS inspector
 - Network analysis
- Cool extensions
 - Yslow – give suggesting for better performance
 - FireRainbow – syntax highlighting
 - FireQuery for jQuery users

Document Object Model

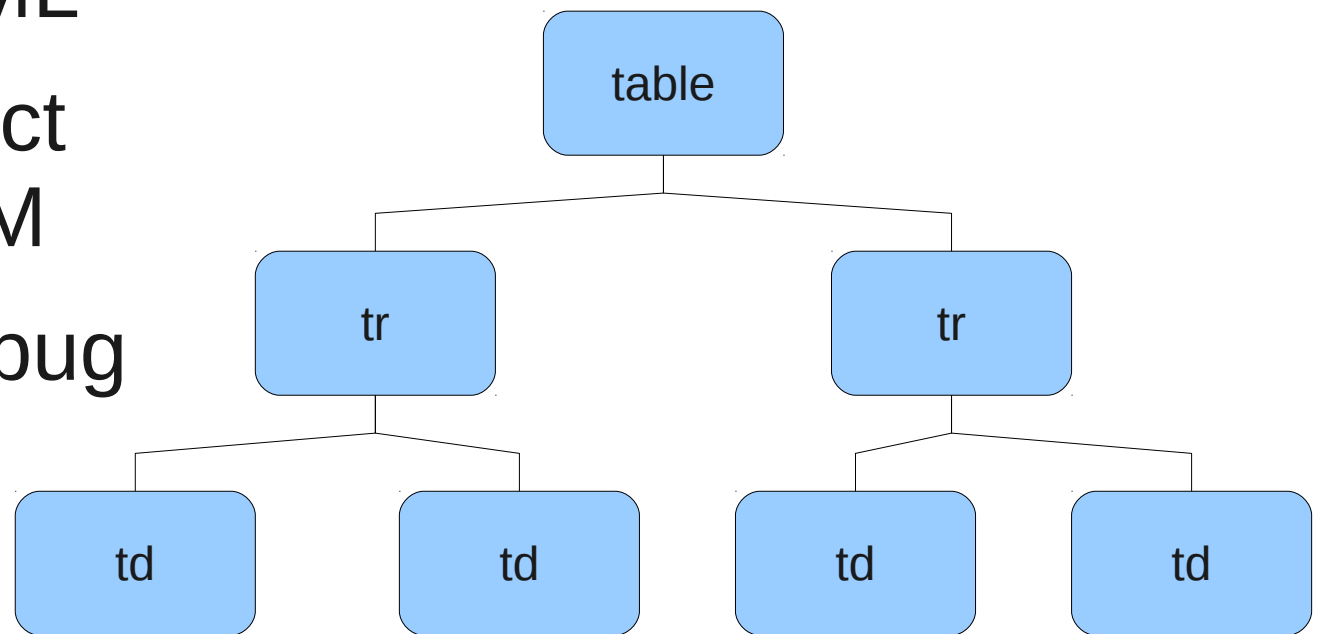
- HTML can be a strict file format
- Browsers read and parse HTML
- Creates a object tree – aka DOM
- Viewed in Firebug

```
<table>
```

```
<tr><td>X</td><td>Y</td></tr>
```

```
<tr><td>A</td><td>B</td></tr>
```

```
</table>
```



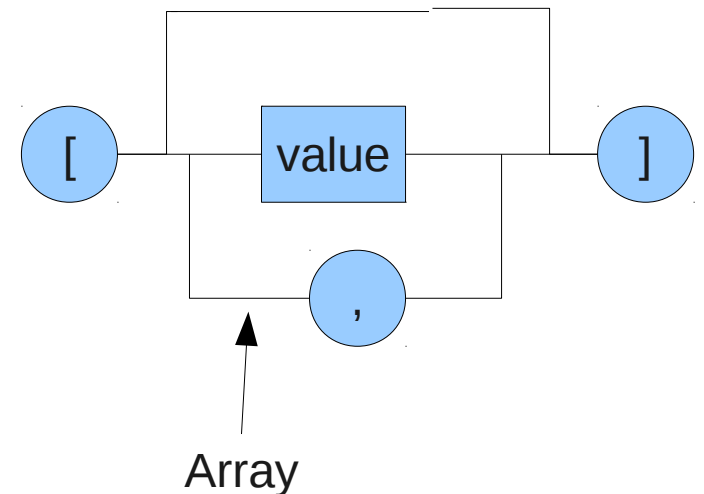
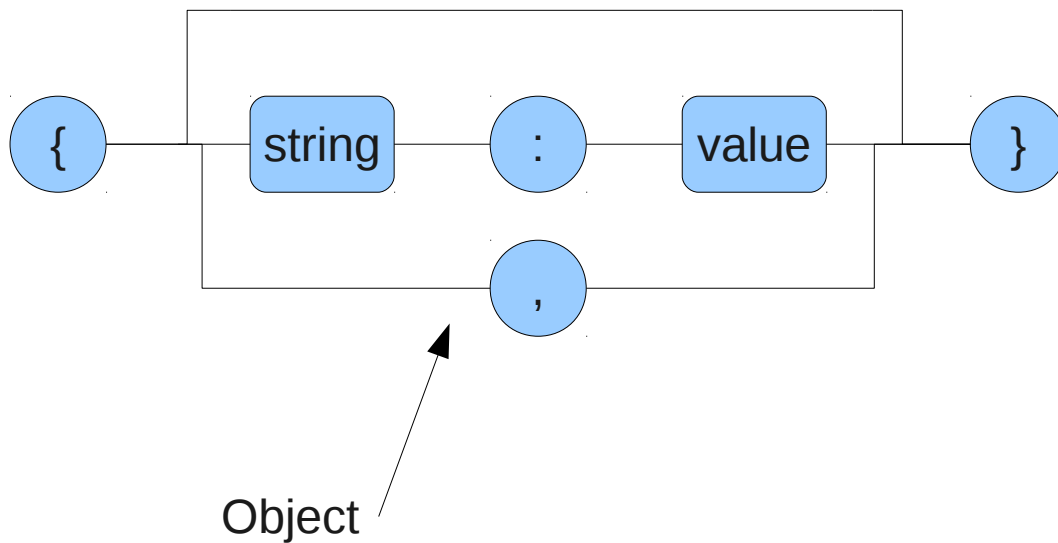
DOM and JavaScript

- JavaScript has a runtime library to perform DOM manipulations
- `document` is the root
- Give relevant HTML elements an ID
- `onClick` property useful
- Rich API
 - `getElementById`, `appendChild`, `removeChild`, `createElement`

Example: dom.html

JavaScript Object Notation (JSON)

- Serialization format for JavaScript objects
 - Text representation of object
- Useful for client/server communication
- Supported by most programming languages



- Browsers have a JSON object
 - `stringify` – create JSON representation
 - `parse` – create an object

Example: `json.js`

- Asynchronous Javascript And XML
- Main concepts
 - Client-side communication with server
 - Update DOM – not reload page
 - Smaller, well defined backend functions
- XMLHttpRequest object is the core technology
- Web applications look and feel like desktop applications

Example: ajax-generic.js, ajax-calc.js, calc.php, calc.html

- Server-side programming in any language
 - Perl, Python, Java, C#
- PHP offers many nice features
 - Parsing of query parameters
 - JSON
 - `json_decode` creates PHP objects
 - `json_encode` serializes PHP objects
 - They are slow – use only for small objects!
 - Access to libraries (MySQL, etc.)

Third party libraries

- Open Source libraries try to make cross browser development easier
- Prototype
 - Early adaptor – basic functionality
 - Scriptaculous (and others) – effects
- jQuery
 - Heavily used today
 - Drupal, Microsoft, and others
- YUI – Yahoo! Library
- Dojo – advanced but hard to learn

- DOM manipulations
- Forms
- Events
- AJAX
- Fluent interface
 - `$(document . body) . css (" background " , " black ")`
- Fancy effects (folding, etc.)
- Cool stuff (drag-n-drop, etc.)



- `$ ()` selects elements based on
 - Style class
 - Element ID
- Create and add new elements
 - `.append` – add DOM elements after
 - `.prepend` – add DOM element before
 - `.html` – add raw HTML
- Delete elements
 - `.remove` – remove elements and it's children
 - `.empty` – remove child nodes
- Style
 - `.css` – sets using CSS

Selectors

- `$ (...)` is the key function in jQuery
- Select DOM elements using CSS selectors
- `$ (...)` returns an array of matching elements

Selector	Description
' #id '	Elements' ID attribute
' .class '	CSS class
' * '	All elements
' [attr="value"] '	Attribute matching
' tag '	Matching HTML tags

Expresion	Description
parent > child	Work on subtree
attr\$="value"	Match at end
attr^="value"	Match at start
attr*="value"	Substring match

Example: selector.html

Useful methods

- Each – iterator using anonymous function
- Append – add an element
- Remove – delete an element
- Empty – delete all element
- Html – get or set raw inner HTML of element
- Val – get or set the value

Effects

- A queue of effects is maintained
 - Effects have optional timing parameter
 - Integer (ms), `slow` (600 ms), `fast` (200 ms)
- Effects can be stopped and dequeued

Methods	Effect
<code>show/hide/toogle</code>	visibility
<code>fadeIn/fadeOut/fadeToogle</code>	Visibility - gradually
<code>slideUp/slideDown/slideToogle</code>	Visibility - rolling
<code>delay</code>	Pause queue execution

- Events trigger callback functions
 - Anonymous functions are ideal candidates
- Method `bind` can add a callback function to any element
 - `click` and `submit` are common events
 - `$(this)` is DOM element as jQuery object
- Method `unbind` removes trigger

- Use a selector to find all input elements
- Method `serialize` is smarter
 - Find input elements
 - Creates a URL string for the values
- Method `serializeArray` returns an array
 - (name, value) pairs
- `val` returns value of first element

Example: form.html

- `$.ajax` is your friend
 - `$.ajax(url, { })`
- It's a remote procedure call/remote method invocation
- Three important options
 - `URL` – address of backend function
 - `data` – the parameters
 - `success` – call back function

Example: `jquery-ajax.html`, `calc2.php`

Drag-n-drop

- Tracking all mouse events
 - Mouse down to grab element
 - Follow mouse movement – pixel by pixel
 - Mouse released to drop element

⇒ Many lines of code

- JQuery simplifies it
 - Define areas (source and destination)
 - Use `draggable` and `droppable`

JavaScript on the server

- Server-side programming
- Based on Google's V8 engine
- Event-driven, non-blocking I/O
- Used by Google, Microsoft, eBay, LinkedIn, Yahoo!
- Modules for
 - TCP listeners, buffers, streams, file I/O, etc.
- Third party modules
 - MySQL, RabbitMQ, ncurses, etc.



- Installation

- `./configure; make ; make install -`
compiles many C++ files!
- Or binary packages (Windows, OS X, some Linux distros)

- The `node` or `nodejs` program is the main program

- Load modules using `require`

```
var foo = require('foo')
```

Modules

- `NODE_PATH` sets the path for global modules
- Either low level C++ modules
 - This is hard work!
- Or written in JavaScript
 - `Export` is used to export functions
 - `Require` is cached
 - And cycles can be detected

Example: `diff.js`, `use-diff.js`

- Objects emit events
 - When a connection is opened (server)
 - When a file is opened (file system)
- Add an event listener to handle events
 - Use `addListener` or `on`
 - And `removeListener` when you don't need it
 - Just like in the browser

Example: `cat.js`

Networking

- Node.js offers various server classes
 - `net` is a general TCP server
 - `http` and `https` are for web servers
 - `cluster` can spawn multiple listeners
- Clients can be written as well
 - `dns` for name look ups
- UDP is also supported through `dgram`

Example: `node-http.js`

- <http://ajaxian.com> is a great news site for Javascript developers
- *Introduktion til Javascript* by K. Geisshirt. Libris, 2010.
- *JavaScript: The Good Parts* by D. Crockford. O'Reilly, 2008
- *Webmaster in a nutshell* by R. Eckstein and S. Spainhour. O'Reilly, 2002
- <http://www.wikipedia.org/Javascript>

- Javascript supports three programming paradigms
 - Procedural
 - Object-oriented
 - Functional
- Under active development
 - Revision of the standard
 - New engines
 - Many frameworks