中山大學
SUN YAT-SEN UNIVERSITY

Lecture 14
Object Oriented JavaScript

SE-805 Web 2.0 Programming (supported by Google)

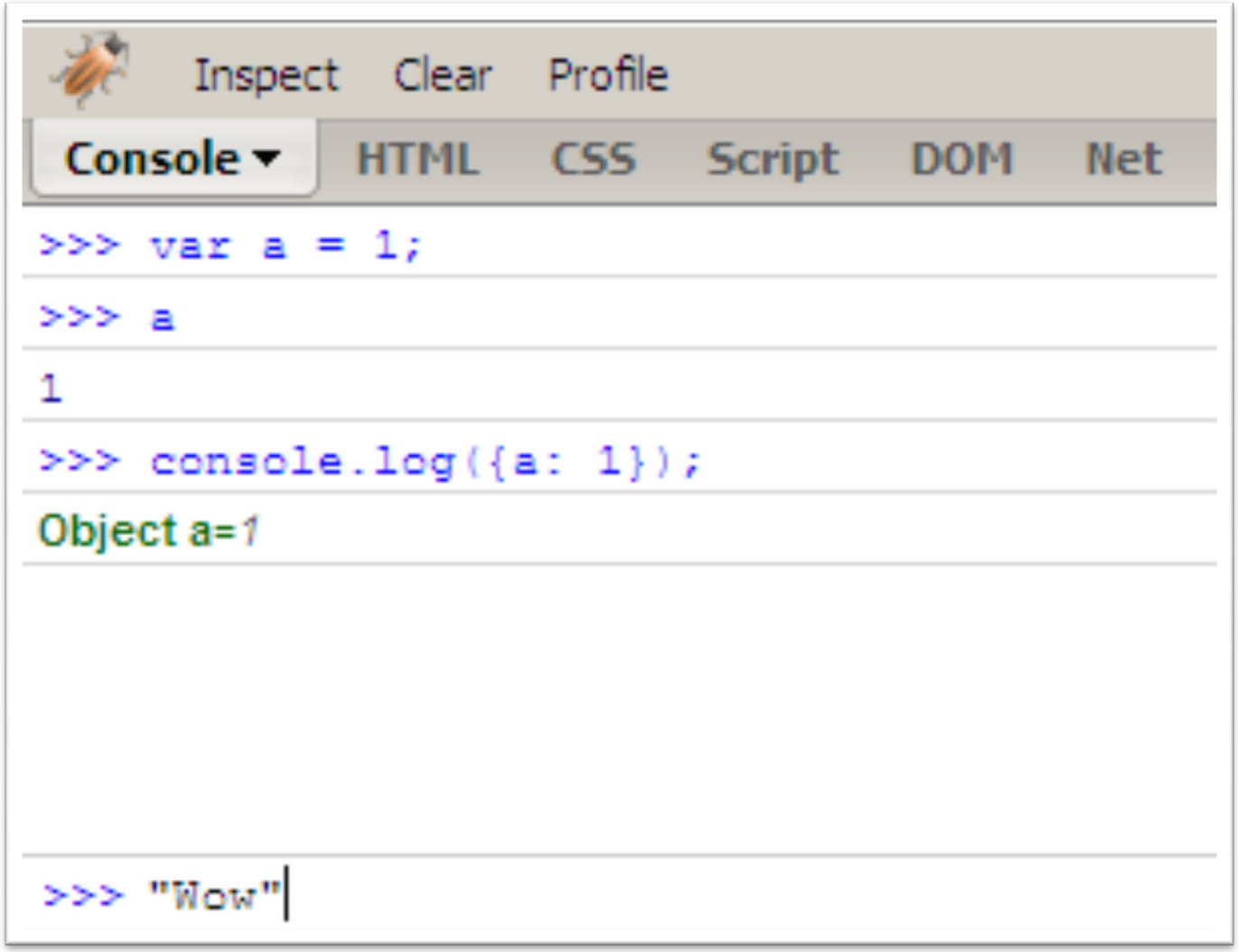
http://my.ss.sysu.edu.cn/courses/web2.0/

School of Software, Sun Yat-sen University

Outline

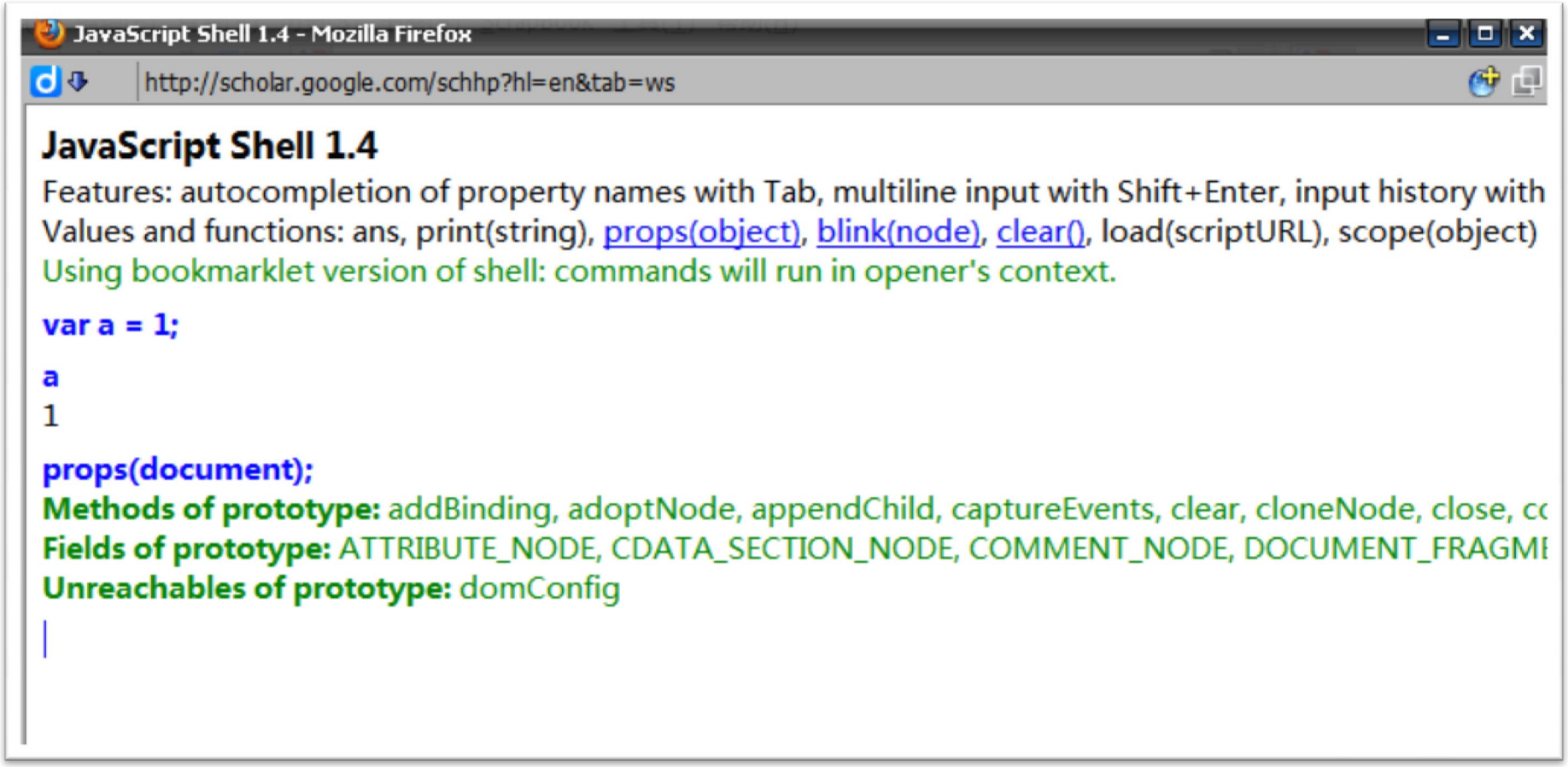
- **Learning Tools**
- Object and Functions
- Constructors and Prototype
- Inheritance
- Polymorphism

Learning Tool – Firebug



Learning Tool – JavaScript Shell

- Using bookmarklet version of shell: commands will run in opener's context
  - https://www.squarefree.com/bookmarklets/webdevel.html



Outline

- Learning Tools
- **Object and Functions**
- Constructors and Prototype
- Inheritance
- Polymorphism

JavaScript != Java

- C-like syntax ✔
- Classes ✘ □
- Data type:
  - Primitive:
    - Number – 1, 3, 1001, 11.12, 2e+3
    - String – "a", "stoyan", "0"
    - Boolean – true | false
    - null
    - undefined
  - Objects: everything else …

Objects

- Every object in fact is a hash table internally (key: value)
- When a property is a function we can call it a method

```
var obj = {};
obj.name = 'my object';
obj.shiny = true;
```

```
var obj = {
  shiny: true,
  isShiny: function() {
    return this.shiny;
  }
};

obj.isShiny(); // true
```

Object Literal Notation

- Key-value pairs
- Comma-delimited
- Wrapped in curly braces

```
{a: 1, b: "test"}
```

Arrays

- Arrays are objects too
- Auto-increment properties
- Some useful methods

```
>>> var a = [1,3,2];
>>> a[0]
1
>>> typeof a
"object"
```

- Array literal notation

```
var array = [
  "Square",
  "brackets",
  "wrap",
  "the",
  "comma-delimited",
  "elements"
];
```

JavaScript Object Notation (JSON)

- Object literals + array literals

```
{"num": 1, "str": "abc",
  "arr": [1,2,3]}
```

- Serialization of Objects, useful in persisting and transporting Objects
- A JSON literal string can be embodied via the eval() function

```
var jsonStr = '{"num": 1, "str": "abc",
  "arr": [1,2,3]}';
obj = eval(jsonStr);
```

Functions

- Functions are objects
  - They have properties
  - They have methods
  - Can be copied, deleted, augmented...
  - Special feature: invokable

```
function say(what) {
  return what;
}
var say = function(what) {
  return what;
};
var say = function say(what) {
  return what;
};
```

Functions are Objects

```
>>> say.length
1
>>> say.name
"boo"
```

```
>>> var tell = say;
>>> tell("doodles")
"doodles"
>>> tell.call(null, "moo!");
"moo!"
```

Return Values

- All functions always return a value
- If a function doesn't return a value explicitly, it returns **undefined**
- Functions can return objects, including other functions

Outline

- Learning Tools
- Object and Functions
- **Constructors and Prototype**
- Inheritance
- Polymorphism

Construction Functions

- When invoked with new, functions return an object known as this
- You can modify this before it's returned
- Naming convention: `MyConstructor`, `myFunction`

```
var Person = function(name) {
  this.name = name;
  this.getName = function() {
    return this.name;
  };
};
var me = new Person("Stoyan");
me.getName(); // "Stoyan"
```

Constructor Property

```
>>> function Person(){};
>>> var jo = new Person();
>>> jo.constructor === Person
true

>>> var o = {};
>>> o.constructor === Object
true
>>> [1,2].constructor === Array
true
```

Built-in Constructors

- Object
- Array
- Function
- RegExp
- Number
- String
- Boolean
- Date
- Error, SyntaxError, ReferenceError…

Conventions

Use this	Not that
`var o = {};`	`var o = new Object();`
`var a = [];`	`var a = new Array();`
`var re = /[a-z]/gmi;`	`var re = new RegExp( '[a-z]', 'gmi');`
`var fn = function(a, b){ return a + b; }`	`var fn = new Function( 'a, b','return a+b');`

Prototype

- prototype is a special property of the function objects
- prototype is **NOT** the JavaScript toolkit we used

```
>>> var boo = function(){};
>>> typeof boo.prototype
"object"
```

- Augmenting the prototype

```
>>> boo.prototype.a = 1;
>>> boo.prototype.sayAh = function(){};
```

- Overwriting the prototype

```
>>> boo.prototype ={a: 1, b: 2};
```

Use of the Prototype

- The prototype is used when a function is called as a constructor

```
var Person = function(name) {
  this.name = name;
};
Person.prototype.say = function(){
  return this.name;
};
```

```
>>> var dude = new Person('dude');
>>> dude.name;
"dude"
>>> dude.say();
"dude"
```

`say()` is a property of the `prototype` object but it behaves as if it's a property of the `dude` object

Own Properties vs. Prototype's

```
>>> dude.hasOwnProperty('name');
true
>>> dude.hasOwnProperty('say');
false
```

- **isPrototypeOf()**

```
>>> Person.prototype.isPrototypeOf(dude);
true
>>> Object.prototype.isPrototypeOf(dude);
true
```

__proto__

- The objects have a secret link to the prototype of the constructor that created them

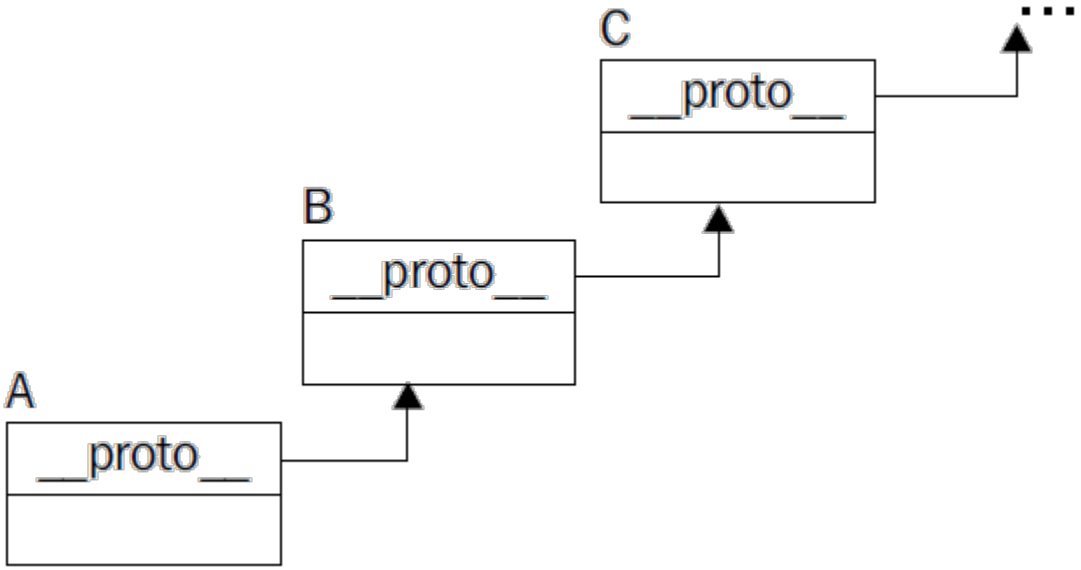
```
>>> dude.__proto__.hasOwnProperty('say')
true

>>> dude.prototype
??? // Trick question

>>> dude.__proto__.__proto__.
  hasOwnProperty('toString')
true
```

__proto__ Chain

- It is a live chain




```
>>> typeof dude.numlegs
"undefined"
>>> Person.prototype.numlegs = 2;
>>> dude.numlegs
2
```

Outline

- Learning Tools
- Object and Functions
- Constructors and Prototype
- **Inheritance**
- Polymorphism

How?

```
function NormalObject() { // Parent Constructor
  this.name = 'normal';
  this.getName = function() {
     return this.name;
  };
}
function PreciousObject(){ // child constructor
  this.shiny = true;
  this.round = true;
}
/*** How can we do this? ***/
var crystal_ball = new PreciousObject();
crystal_ball.name = 'Crystal Ball.';

crystal_ball.round; // true
crystal_ball.getName(); // "Crystal Ball."
```

Object Inheritance by Copying

```
// Two Objects
var shiny = {
  shiny: true,
  round: true
};
var normal = {
  name: 'name me',
  getName: function() {
    return this.name;
  }
};
```

```
// Inheritance tool function
function extend(parent,
   child){
  for (var i in parent) {
    child[i] = parent[i];
  }
}

// Inheritance by copying
extend(normal, shiny);
shiny.getName(); // "name me”
```

Prototypal Inheritance

- Beget object

```
function object(o) {
  function F(){}
  F.prototype = o;
  return new F();
}
```

```
>>> var parent = {a:
  1};
>>> var child =
  object(parent);
>>> child.a;
1
>>>
  child.hasOwnProperty
  (a);
false
```

Outline

- Learning Tools
- Object and Functions
- Constructors and Prototype
- Inheritance
- **Polymorphism**

Is the JavaScript an OOP Language?

- **Definitely sure**!
- Object Oriented **NOT** Class Oriented
  - Encapsulation
  - Inheritance
  - Polymorphism -- Because JavaScript is a dynamic language, polymorphism is quite easy and very common.
  - Two common types of polymorphism:
    - Runtime Replacement
    - Loadtime Branching
- And it is more dynamic then Java and C++, these compiling languages

Loadtime Branching

```
var getXHR = function () {
  if (window.XMLHttpRequest) {
    return function () {
      // Return a standard XHR instance
    };
  }
  else {
    return function () {
      // Return an Explorer XHR instance
    };
  }
}(); // Note: parents trigger self-invocation
```

Runtime Replacement

```
var documentListFactory = function () {
  var out = []; // Just a simple array
  // Override the default .push() method
  out.push = function (document) {
    Array.prototype.push.call(out, {
      document  : document,
      timestamp : new Date().getTime()
    });
  };
  return out;
};
```

Summary

- Learning Tools
  - Firebug, JavaScript Shell
- Object and Functions
  - JavaScript != Java
  - Object literal, Array literal, JSON
  - Functions: objects, invokable, return value
- Constructors and Prototype
  - Constructor functions, constructor property
  - Built-in constructors, conventions
  - Prototype, __proto__ chain
- Inheritance
  - By copying, prototypal
- Polymorphism
  - Loadtime branching, runtime replacement

Exercises

- Write a JavaScript snippets in Firebug console defining classes of Employee, Manager, and Secretary
  - Each Employee has a name and a salary
  - Each Manager is an Employee, and manages a group of other Employees
  - Each Secretary is an Employee, and works for a Manager
- add methods to these classes
  - Each Employee has a show() method returns her name and salary as a string
  - Each Manager has a getInferiors() method returns her inferiors
  - Each Secretary has a getSuperior() method returns her boss
- Inheritances in two different ways, copying and prototype

Further Readings

- Introduction of JavaScript
  http://en.wikipedia.org/wiki/JavaScript
- W3Schools JavaScript tutorial
  http://www.w3schools.com/js/default.asp
- Mozilla Developer Center JavaScript documentation
  https://developer.mozilla.org/en/javascript
- Object Oriented Programming in JavaScript by Mike Koss
  http://mckoss.com/jscript/object.htm
- JavaScript Object-Oriented Programming Part 1
  http://articles.sitepoint.com/article/oriented-programming-1
- JavaScript Object-Oriented Programming Part 2
  http://articles.sitepoint.com/article/oriented-programming-2
-

Thank you!