JavaScript(Object Oriented Concepts)

Authored by :Asfiya Khan Presented by : Asfiya Khan

Agenda

- Strict Mode
- Variable Hoisting
- Objects
- Scope
- Functions
- Closures
- Inheritance

Strict Mode

- Strict mode makes several changes to normal JavaScript semantics.
- First, strict mode eliminates some JavaScript silent errors by changing them to throw errors.
- Second, strict mode fixes mistakes that make it difficult for JavaScript engines to perform optimizations: strict mode code can sometimes be made to run faster than identical code that's not strict mode.
- Third, strict mode prohibits some syntax likely to be defined in future versions of ECMAScript.

Variable hoisting

- A variable can appear to be used before it's declared, is called "hoisting".

  Eg:

```
x = 2
var x;

Is implicitly understood as:
var x;
 x = 2;
```

What is an Object

- An object is an unordered list of primitive data that is stored as a series of name-value pairs.
- Each item in the list is called a *property*. For eg:

```
var myFirstObject = {firstName: "Richard", favoriteAuthor: "Conrad"};
```

- Property names can be a string or a number, but if the property name is a number, it has to be accessed with the bracket notation.

Object creation methods

1. Object Literals
2. Object using Constructor
3. Using Object.create()

Object Literal

```
var obj = { property_1: value_1, //
                 property_# may be an identifier... 2: value_2,
                };
```

Eg:

```
  var dog={name: ‘Fluffy’, color: ‘White’ };
```

Cybage® Delivering Value Scientifically

Using a constructor function

Alternatively, you can create an object with these two steps:

1. Define the object type by writing a constructor function.

2. Create an instance of the object with new.

Using the Object.create method

1. Objects can also be created using the Object.create() method.

2. This method can be very useful, because it allows you to choose the prototype object for the object you want to create.

Additional Information on Properties.

Scopes

There are two scope in JS:

1. Local Scope
2. Global Scope

Functions

- Every function is a Function object.
- A function without a return statement will return a default value.
- In the case of a constructor called with the new keyword, the default value is the value of its this parameter.
- For all other functions, the default return value is undefined.

Closure

- A **closure** is a function together with a *referencing environment* for the non-local variables of that function.
- Its like keeping a copy of all the local variables, just as they were when a function exited.
- A new set of local variables is kept every time a function with a closure is called .

Inheritance

- Inheritance is a way to create a class as a specialized version of one or more classes.
- JavaScript only supports single inheritance.
- Inheritance is achieved this by assigning an instance of the parent class to the child class, and then specializing it.

Prototype

<u>Prototype is an Object that exist on every function in JavaScript.</u>

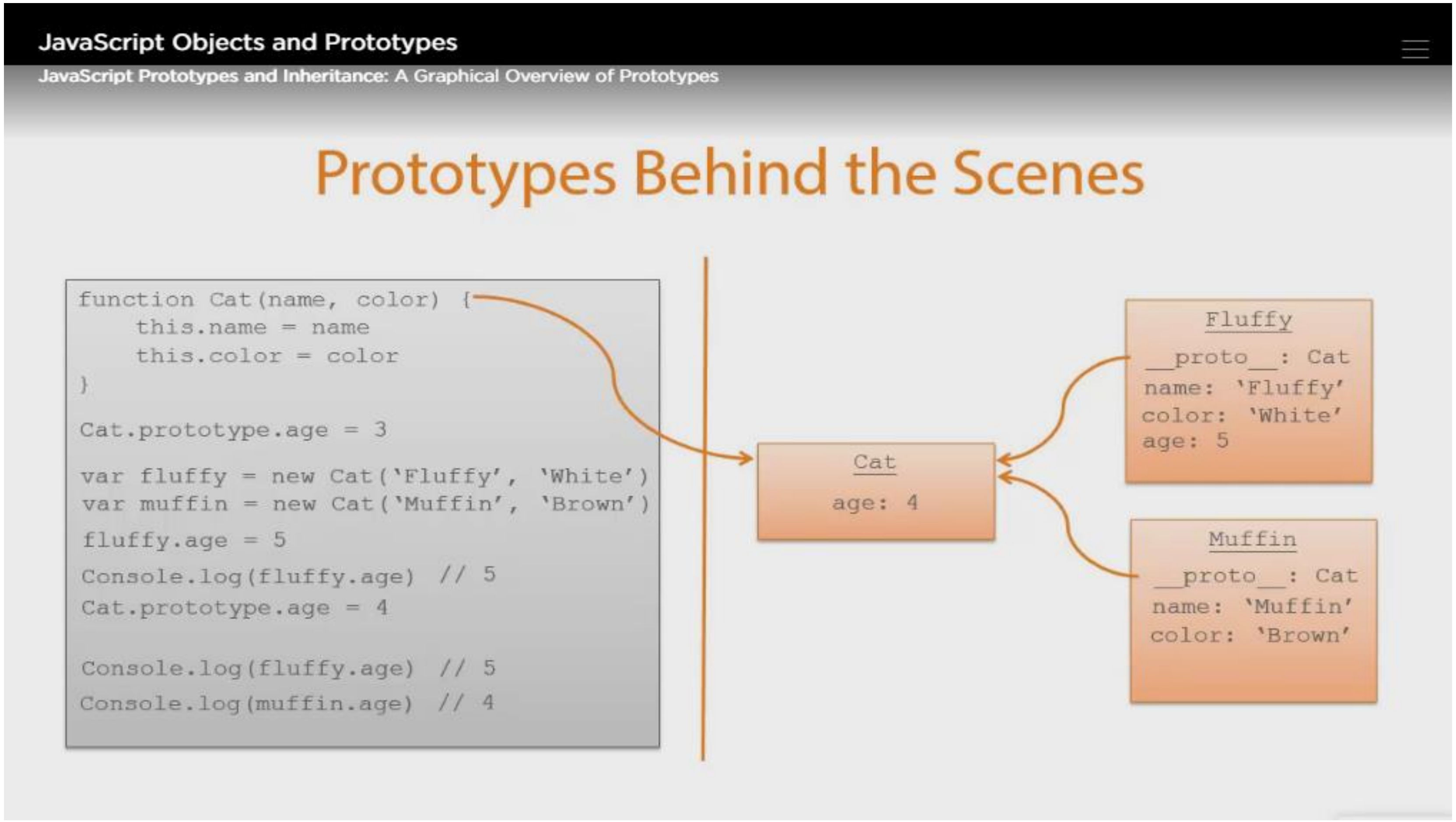
Function's Prototype

A function's prototype is an Object instance that will become the prototype for all objects created using this function as a constructor.

Object Prototype

Its an Object instance from which the object is inherited.

Prototypal Inheritance


JavaScript Objects and Prototypes

JavaScript Prototypes and Inheritance: A Graphical Overview of Prototypes

Prototypes Behind the Scenes

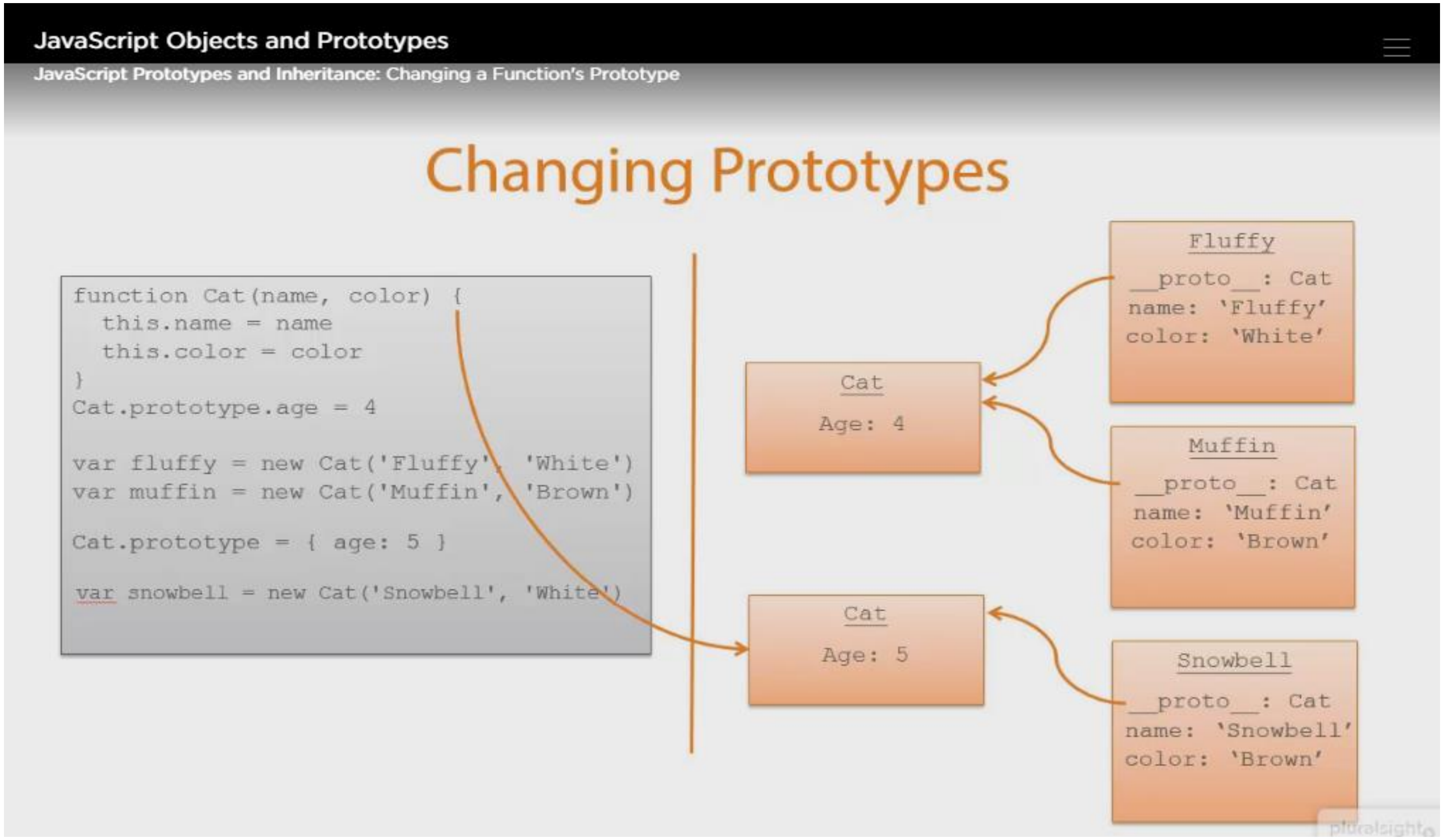
```
function Cat(name, color) {
    this.name = name
    this.color = color
}

Cat.prototype.age = 3

var fluffy = new Cat('Fluffy', 'White')
var muffin = new Cat('Muffin', 'Brown')
fluffy.age = 5
Console.log(fluffy.age)  // 5
Cat.prototype.age = 4

Console.log(fluffy.age)  // 5
Console.log(muffin.age)  // 4
```

```
Cat
age: 4
```

```
Fluffy
__proto__: Cat
name: 'Fluffy'
color: 'White'
age: 5
```

```
Muffin
__proto__: Cat
name: 'Muffin'
color: 'Brown'
```

Changing Prototype



JavaScript Objects and Prototypes

JavaScript Prototypes and Inheritance: Changing a Function's Prototype

Changing Prototypes

```
function Cat(name, color) {
  this.name = name
  this.color = color
}
Cat.prototype.age = 4

var fluffy = new Cat('Fluffy', 'White')
var muffin = new Cat('Muffin', 'Brown')

Cat.prototype = { age: 5 }

var snowbell = new Cat('Snowbell', 'White')
```

JavaScript Patterns

Few Common JS Patterns are:

- Module Pattern
- Revealing Module Pattern
- Singleton Pattern
- Observer Pattern
- Mediator Pattern
- Prototype Pattern
- Façade Pattern
- Factory Pattern

Singleton Pattern

- This pattern restricts instantiation of an object to a single reference.
- Delayed initialization that prevents instantiation until required
- Single Point of Access.

Revealing Module Pattern

- All methods and variables are kept private until they are explicitly exposed.
- They are exposed by an object literal returned by the closure from which it's defined.
- Advantage is that it support private data.

Cybage® Delivering Value Scientifically

Any Questions?

Thank you!