

# Javascript nâng cao

Nguyễn Văn Mạnh

# Nội dung chương trình

- Javascript Object
- Function
- Closures
- Inheritance

# Javascript Object

- Là một danh sách các dữ liệu nguyên thủy, không theo thứ tự.
- Được lưu trữ theo từng cặp name-value
- Các hàm (function) được gọi là các phương thức (methods)
- Tên thuộc tính (Property) có thể là string hoặc number.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

# Tạo đối tượng trong JavaScript

- Tạo object sử dụng **object literal**.
- Tạo object sử dụng từ khóa **new**.
- Tạo object sử dụng **function**.

# Sử dụng Object Literal

- Có thể khai báo các thuộc tính và phương thức của Object trong 1 dòng hoặc nhiều dòng

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

```
var person = {  
    firstName:"John",  
    lastName:"Doe",  
    age:50,  
    eyeColor:"blue"  
};
```

# Sử dụng từ khóa new để tạo đối tượng

- Có thể khởi tạo một đối tượng với từ khóa new:

```
var person = new Object();  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";
```

# Sử dụng function tạo đối tượng

```
function Employee(name, age) {  
    this.name = name;  
    this.age = age;  
    this.go = function () { /*Go action*/ };  
};  
var employee = new Employee();
```

# Các thuộc tính của đối tượng

- Dot notation
  - `obj.lastName = "Smith";`
- Bracket notation
  - `obj["lastName"] = "Smith";`
- Enumerating properties

```
for (prop in inobj) {  
    • alert(prop + ": " + obj[prop]);  
}
```



# Functions

- Các function trong javascript là một first-class: tức là không bị giới hạn về cách tạo và sử dụng.
  - Đặc điểm first-class:
    - Được lưu trữ trong các biến.
    - Được chuyển thành các đối số cho các hàm.
    - Được tạo ra trong các hàm và trả về từ các hàm.
- Các function có thể được tạo ẩn danh bất cứ lúc nào
- Các function có thể được truyền dưới dạng tham số

# Định nghĩa một hàm (function)

```
function canFly() { return true; }
```

```
var canFly = function () { return true; };
```

```
window.canFly = function () { return true; };
```

# Location of function

```
var canFly = function () { return true; };
```

True

```
window.isDeadly = function () { return true; };
```

```
alert(isNimble() && canFly() && isDeadly());
```

```
function isNimble() { return true; }
```

```
alert(canFly() && isDeadly());
```

Error

```
var canFly = function () { return true; };
```

```
window.isDeadly = function () { return true; };
```

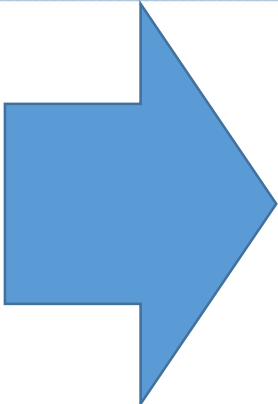
# Function đệ quy

```
function yell(n) {  
    return n > 0 ? yell(n - 1) + "a" : "hiy";  
}  
alert(yell(4));
```

# Function đệ quy trong một đối tượng

```
var ninja = {  
    yell: function (n) {  
        return n > 0 ? ninja.yell(n - 1) + "a" :  
        "hiy";  
    }  
};  
var samurai = { yell: ninja.yell };  
var ninja = {};  
alert(samurai.yell(4));
```

Error

- 
- Tìm hiểu thêm
    - Các sử dụng 'this'
    - Cách sử dụng hàm ẩn danh.
    - arguments.callee

# Functions as Objects

- Trong JavaScript các hàm (function) hoạt động giống như một đối tượng (object)
  - Có thể gán thuộc tính (property) cho một hàm (function)

```
function isPrime(num) {  
    if (isPrime.answers[num] != null)  
        return isPrime.answers[num];  
    var prime = num != 1;  
    for (var i = 2; i < num; i++) {  
        if (num % i == 0) {  
            prime = false;  
            break;  
        }  
    }  
    return isPrime.answers[num] = prime;  
}  
isPrime.answers = {};
```

# .call() vs .apply()

- Điều nằm trong function prototype nên chỉ có function mới có thể gọi được:
  - **.call()**: truyền lần lượt các tham số.
  - **.apply()**: truyền một mảng tham số.
- Cú pháp:

```
call()
Function.prototype.call(thisArg[, arg1[, arg2, ...]])
apply()
Function.prototype.apply(thisArg, argArray)
```

# Variable Arguments

- Function trong JS có thể chấp nhận bất kỳ một số lượng đối số nào.
- Luôn truy cập được tất cả các đối số thông qua biến

*“arguments”*

```
function Sum() {  
    var total = 0;  
    for (var i = 0; i < arguments.length; i++) {  
        total += arguments[i];  
    }  
    return total;  
}
```

```
alert(Sum(1, 2, 3));
```



# Function overloading

- Thuộc tính “**length**” của function tương ứng với số lượng đối số của hàm đó:

```
function Ninjas(){
    addMethod(this, "find", function(name){});
    addMethod(this, "find", function(first, last){});
}

function addMethod(object, name, fn) {
    var old = object[name];
    object[name] = function () {
        if (fn.length == arguments.length)
            return fn.apply(this, arguments)
        else if (typeof old == 'function')
            return old.apply(this, arguments);
    };
}
```

# JavaScript Closures

- Closure là hàm đi kèm với môi trường mà nó tham chiếu đến.
  - Nó có thể truy cập tới scope (phạm vi) riêng của chính nó.
  - Nó có thể truy cập tới các biến bên ngoài hàm.
  - Nó có thể truy cập tới biến toàn cục.
  - Hàm bên trong cũng có thể truy cập tới các tham số của hàm bên ngoài. (but “arguments”).

# Ví dụ về Closures

```
var a = 5;  
function runMe(a) {  
    alert(a); //?
```

← 6

```
    function innerRun() {  
        alert(b); //?  
        alert(c); //?
```

← 7

← undefined

```
    }  
    var b = 7;  
    innerRun();  
    var c = 8;  
    alert(d); //?  
}  
var d = 9;  
runMe(6);
```

← 9

```
$(document).ready(function () {  
    var count = 0;  
    $("#btn").click(function () {  
        alert("Click count: " +  
            (++count).toString());  
    });  
});
```

# Javascript Closures

- Closures có thể truy cập các biến ngoài hàm ngay sau khi hàm ngoài trả về.

```
function Count() {  
  var count = 0;  
  function innerCount() {  
    alert( ++count);  
  }  
  return innerCount;  
}
```

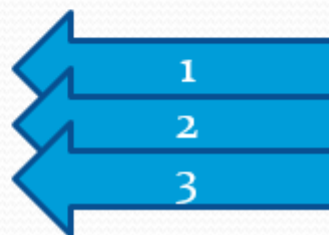
*“Closures store  
references to outer  
function’s variables”*

```
var objCount = Count();
```

```
objCount(); //?
```

```
objCount(); //?
```

```
objCount(); //?
```



# Closure issue

```
$(document).ready(function () {  
    for (var i = 1; i <= 3; i++) {  
        $("#btn" + i).click(function () {  
            alert("Button " + i + " was clicked");  
        });  
    }  
});
```

# (function(){})( )

- Temporary Scope
- Immediately invoke function expression
- Khắc phục được điểm yếu của closure (Closure issue)

```
$(document).ready(function ( ) {  
    for (var i = 1; i <= 3; i++) {  
        (function (i) {  
            $("#btn" + i).click(function ( ) {  
                alert("Button " + i + " was clicked");  
            });  
        })(i);  
    }  
});
```

# Function prototype

- Mọi function trong JS đều có một thuộc tính “prototype”
- Thêm các phương thức hoặc thuộc tính vào thuộc tính “prototype” để các phương thức hoặc thuộc tính đó có thể có trong thể hiện của hàm đó.

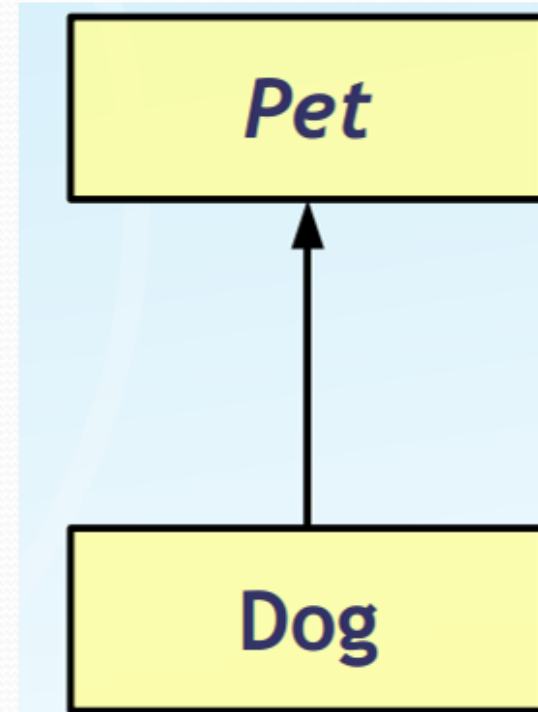
```
function Pet(name) {  
    this.name = name;  
}  
  
var p = new Pet("Micky");  
  
Pet.prototype.getName = function () {  
    return this.name;  
}  
alert(p.getName());
```



Micky

# Inheritance (kế thừa)

```
function Pet(name) {  
    this.name = name;  
}  
  
function Dog(name) {  
    Pet.call(this, name);  
}  
  
Dog.prototype = new Pet();
```





THANK YOU