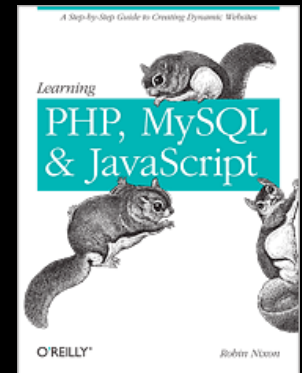


JavaScript Arrays, Objects, and Control Flow

Chapter 16
Dr. Charles Severance

To be used in association with the book:
PHP, MySQL, and JavaScript by Robin Nixon



open.michigan

Unless otherwise noted, the content of this course material is licensed under a Creative Commons Attribution 3.0 License.

<http://creativecommons.org/licenses/by/3.0/>.

Copyright 2011, Charles Severance



Definitions



- **Class** - a template - Dog
- **Method or Message** - A defined capability of a class - bark()
- **Object or Instance** - A particular instance of a class - Lassie

Terminology: Class



Defines the abstract characteristics of a thing (object), including the thing's characteristics (its attributes, fields or properties) and the thing's behaviors (the things it can do, or methods, operations or features). One might say that a **class** is a **blueprint** or factory that describes the nature of something. For example, the **class** Dog would consist of traits shared by all dogs, such as breed and fur color (characteristics), and the ability to bark and sit (behaviors).

http://en.wikipedia.org/wiki/Object-oriented_programming

Terminology: Class



A pattern (exemplar) of a **class**. The **class** of Dog defines all possible dogs by listing the characteristics and behaviors they can have; the object Lassie is one particular dog, with particular versions of the characteristics. A Dog has fur; Lassie has brown-and-white fur.

http://en.wikipedia.org/wiki/Object-oriented_programming

Terminology: Instance



One can have an **instance** of a class or a particular object. The **instance** is the actual object created at runtime. In programmer jargon, the Lassie object is an **instance** of the Dog class. The set of values of the attributes of a particular **object** is called its state. The **object** consists of state and the behavior that's defined in the object's class.

Object and Instance are often used interchangeably.

http://en.wikipedia.org/wiki/Object-oriented_programming

Terminology: Method



An object's abilities. In language, **methods** are verbs. Lassie, being a Dog, has the ability to bark. So bark() is one of Lassie's methods. She may have other **methods** as well, for example sit() or eat() or walk() or save_timmy(). Within the program, using a **method** usually affects only one particular object; all Dogs can bark, but you need only one particular dog to do the barking

Method and Message are often used interchangeably.

http://en.wikipedia.org/wiki/Object-oriented_programming

Objects in JavaScript

- An object groups data together along with functions needed to manipulate it.
- A "class" is a template that defines the shape/structure for the object

Objects in JavaScript

- The OO Pattern in JavaScript is a little different
- The function is indeed a store and reuse pattern
- The function keyword returns a value which is the function itself - it makes a function!

A Sample Class



class is a reserved word.

Each PartyAnimal object has a bit of code.

Tell the object to run the party() code.

```
class PartyAnimal:
```

```
    x = 0
```

```
    def party(self) :
```

```
        self.x = self.x + 1
```

```
        print "So far",self.x
```

```
an = PartyAnimal()
```

```
an.party()
```

```
an.party()
```

```
an.party()
```

This is the template for making PartyAnimal objects.

Each PartyAnimal object has a bit of data.

Create a PartyAnimal object.

PartyAnimal.party(an)

run party() *within* the object an

```
function PartyAnimal() {  
  this.x = 0;  
  this.party = function () {  
    this.x = this.x + 1;  
    console.log("So far " + this.x);  
  }  
}
```

This is the template for making PartyAnimal objects.

Each PartyAnimal object has a bit of data.

Each PartyAnimal object has a bit of code.

```
an = new PartyAnimal();
```

Create a PartyAnimal object.

```
an.party();  
an.party();  
an.party();
```

Tell the object to run the party() code.

```
function PartyAnimal() {  
  this.x = 0;  
  this.party = function () {  
    this.x = this.x + 1;  
    console.log("So far " + this.x);  
  }  
}
```

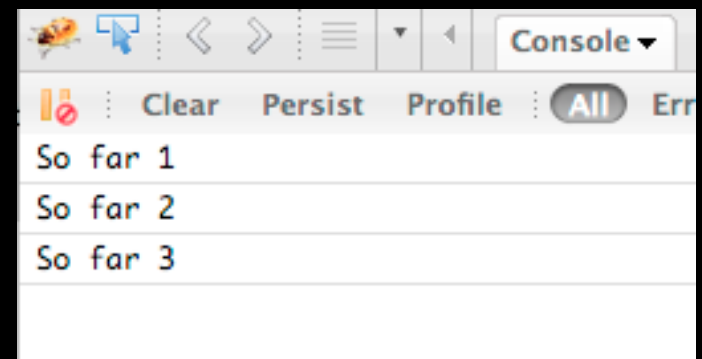
```
an = new PartyAnimal();
```

```
an.party();  
an.party();  
an.party();
```

an

x:

party()



Object Life Cycle

[http://en.wikipedia.org/wiki/Constructor_\(computer_science\)](http://en.wikipedia.org/wiki/Constructor_(computer_science))

Object Life Cycle

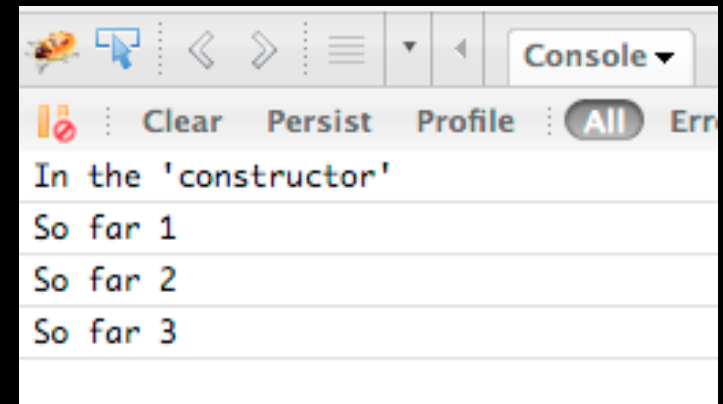
- Objects are created, used and discarded
- Constructors are implicit in JavaScript - natural
 - A **constructor** in a class is a special block of statements called when an object is created
- Destructors are not provided by JavaScript

[http://en.wikipedia.org/wiki/Constructor_\(computer_science\)](http://en.wikipedia.org/wiki/Constructor_(computer_science))

```
function PartyAnimal() {  
  this.x = 0;  
  console.log("In the 'constructor'");  
  this.party = function () {  
    this.x = this.x + 1;  
    console.log("So far "+this.x);  
  }  
}
```

```
an = new PartyAnimal();
```

```
an.party();  
an.party();  
an.party();
```



Many Instances

- We can create **lots of objects** - the class is the template for the object
- We can store each **distinct object** in its own variable
- We call this having multiple **instances** of the same class
- Each **instance** has its own copy of the **instance variables**

```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, nam):
        self.name = nam
        print self.name, "constructed"

    def party(self) :
        self.x = self.x + 1
        print self.name, "party count", self.x

s = PartyAnimal("Sally")
s.party()

j = PartyAnimal("Jim")
j.party()
s.party()
```

Python Throwback

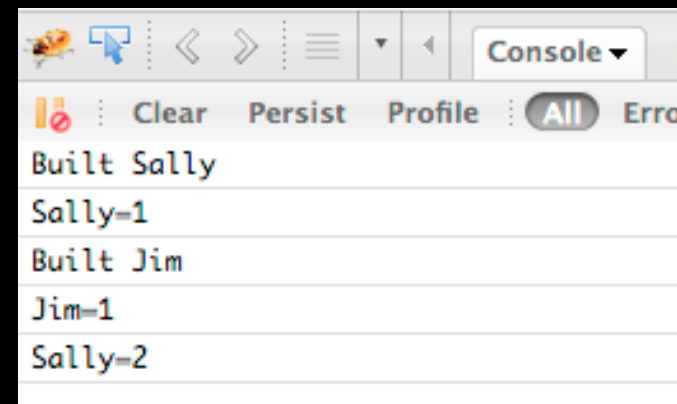
Constructors can have additional parameters. These can be used to setup instance variables for the particular instance of the class (i.e. for the particular object).

```
function PartyAnimal(nam) {  
  this.x = 0;  
  this.name = nam;  
  console.log("Built "+nam);  
  this.party = function () {  
    this.x = this.x + 1;  
    console.log(nam+"="+this.x);  
  }  
}
```

```
s = new PartyAnimal("Sally");  
s.party();
```

```
j = new PartyAnimal("Jim");  
j.party();  
s.party();
```

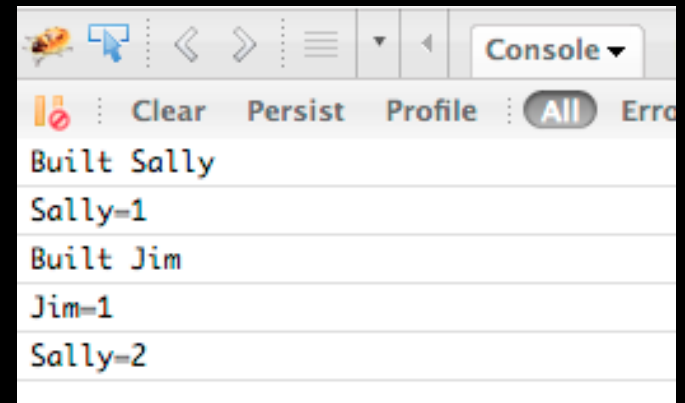
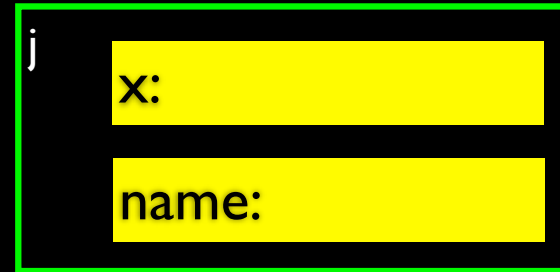
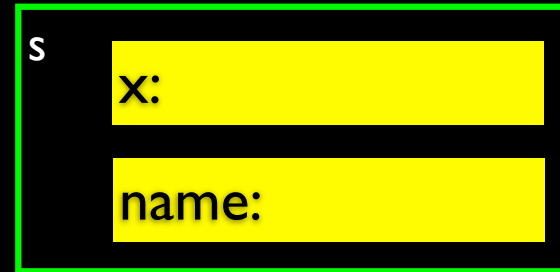
Constructors can have additional **parameters**. These can be used to setup **instance variables** for the particular instance of the class (i.e. for the particular object).



```
function PartyAnimal(nam) {  
  this.x = 0;  
  this.name = nam;  
  console.log("Built "+nam);  
  this.party = function () {  
    this.x = this.x + 1;  
    console.log(nam+"="+this.x);  
  }  
}
```

```
s = new PartyAnimal("Sally");  
s.party();
```

```
j = new PartyAnimal("Jim");  
j.party();  
s.party();
```



Definitions

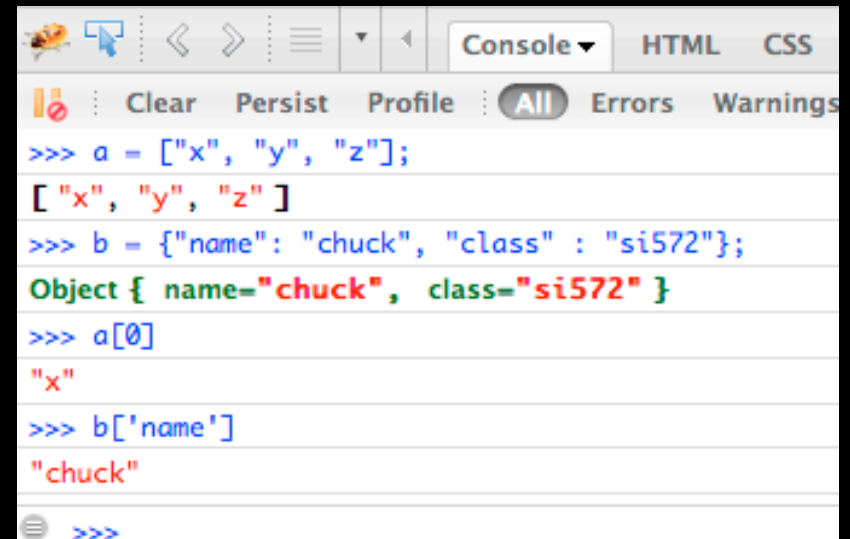


- **Class** - a template - Dog
- **Method or Message** - A defined capability of a class - bark()
- **Object or Instance** - A particular instance of a class - Lassie
- **Constructor** - A method which is called when the instance / object is created

Arrays

Arrays

- JavaScript supports both linear arrays and associative structures, but the associative structures are actually objects



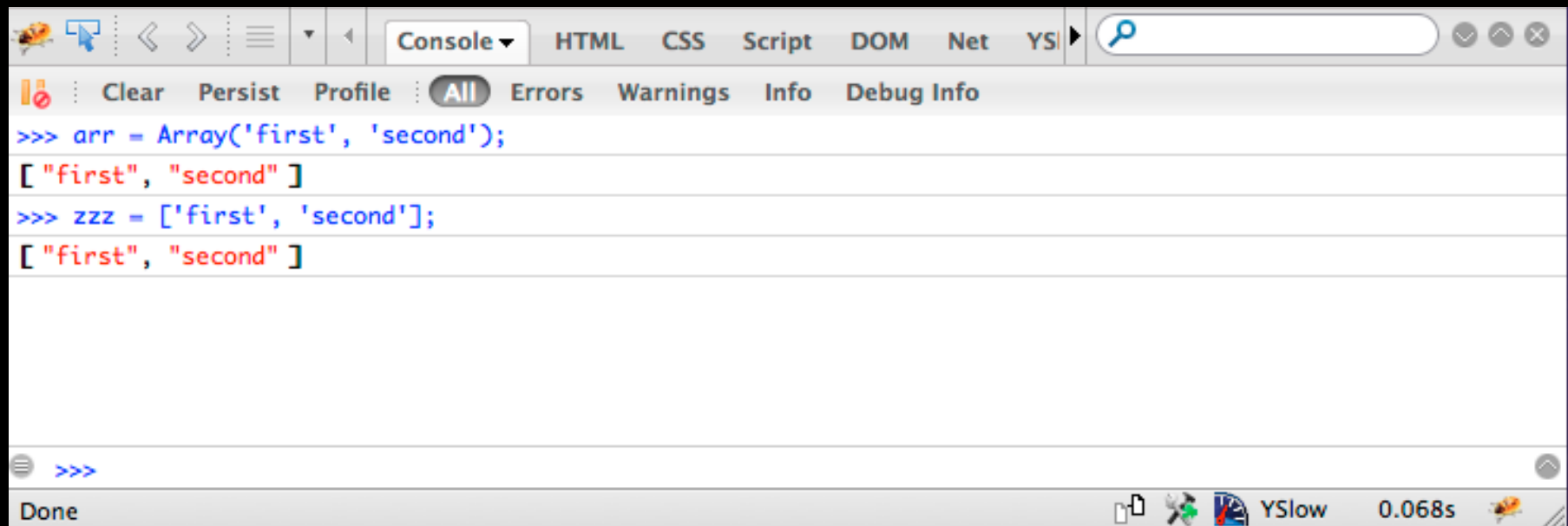
```
>>> a = ["x", "y", "z"];
["x", "y", "z"]
>>> b = {"name": "chuck", "class": "si572"};
Object { name="chuck", class="si572" }
>>> a[0]
"x"
>>> b['name']
"chuck"
>>>
```

Linear Arrays

```
Console HTML
Clear Persist Profile All Errors Wa
>>> arr = Array()
[ ]
>>> arr.push('first');
1
>>> arr.push('second')
2
>>> arr
[ "first", "second" ]
>>>
```

```
Console HTML
Clear Persist Profile All Errors Wa
>>> arr = Array()
[ ]
>>> arr[0] = 'first';
"first"
>>> arr[1] = 'second';
"second"
>>> arr
[ "first", "second" ]
>>>
```

Array Constructor



The screenshot shows a browser's developer console with the following content:

```
>>> arr = Array('first', 'second');  
[ "first", "second" ]  
>>> zzz = ['first', 'second'];  
[ "first", "second" ]
```

The console interface includes a toolbar with icons for search, back, forward, and refresh. Below the toolbar are tabs for 'Console', 'HTML', 'CSS', 'Script', 'DOM', 'Net', and 'YSlow'. A search bar is located to the right of the tabs. Below the search bar are buttons for 'Clear', 'Persist', 'Profile', and a dropdown menu currently set to 'All'. Further right are buttons for 'Errors', 'Warnings', 'Info', and 'Debug Info'. At the bottom of the console, there is a 'Done' status indicator on the left and a 'YSlow' performance tool indicator on the right, showing a score of 0.068s.

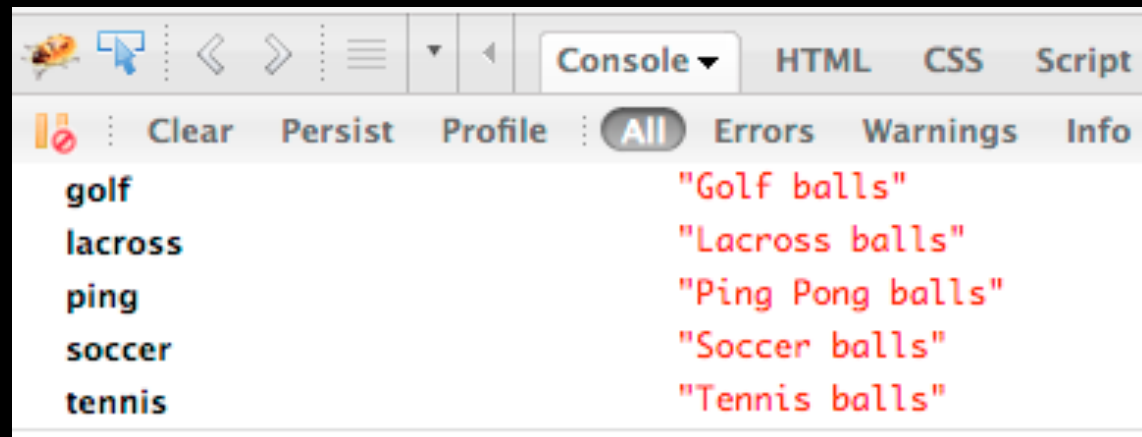
Associative ~~Arrays~~ Objects

- JavaScript Associative Arrays are actually objects with member variables
- They can be accessed with either associative array syntax or object syntax

```
balls = {"golf": "Golf balls",  
        "tennis": "Tennis balls",  
        "ping": "Ping Pong balls"};
```

```
balls.soccer = "Soccer balls";  
balls['lacross'] = "Lacross balls";
```

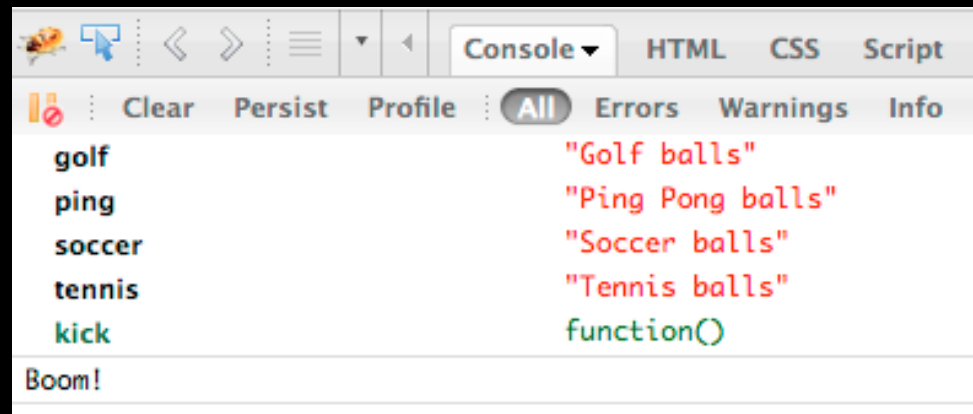
```
console.dir(balls);
```



```
balls = {"golf": "Golf balls",  
        "tennis": "Tennis balls",  
        "ping": "Ping Pong balls"};  
balls.soccer = "Soccer balls";
```

```
balls.kick = function () {  
    console.log('Boom!');  
}
```

```
console.dir(balls);  
balls.kick();
```



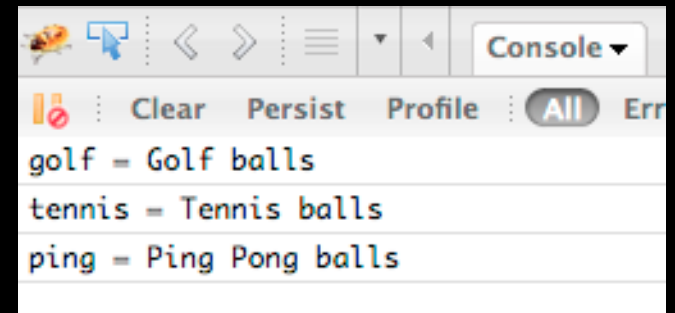
Control Structures Lite...

Control Structures

- We use curly braces for control structure and whitespace / line ends do not matter
- **If** statements are as you would expect
- **While** loops are as you would expect
- Counted **for** loops are as you would expect
- In loops, **break** and **continue** are as you would expect

Definite Loops (for)

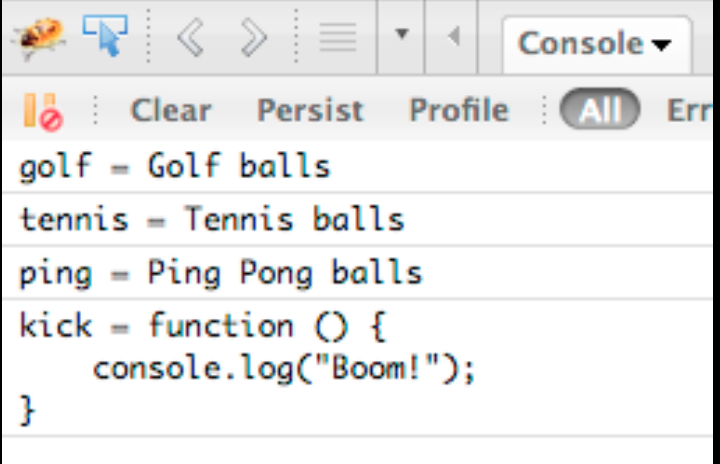
```
balls = {"golf": "Golf balls",  
        "tennis": "Tennis balls",  
        "ping": "Ping Pong balls"};  
  
for (ball in balls) {  
    console.log(ball+' = '+balls[ball]);  
}
```



```
balls = {"golf": "Golf balls",
        "tennis": "Tennis balls",
        "ping": "Ping Pong balls"};

balls.kick = function () {
    console.log('Boom!');
}

for (ball in balls) {
    console.log(ball+' = '+balls[ball]);
}
```

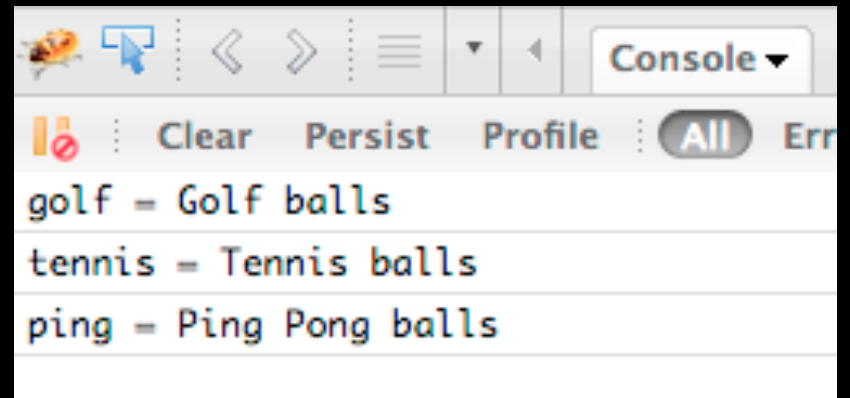


```
Console
Clear Persist Profile All Err
golf = Golf balls
tennis = Tennis balls
ping = Ping Pong balls
kick = function () {
    console.log("Boom!");
}
```

```
balls = {"golf": "Golf balls",  
        "tennis": "Tennis balls",  
        "ping": "Ping Pong balls"};
```

```
balls.kick = function () {  
    console.log('Boom!');  
}
```

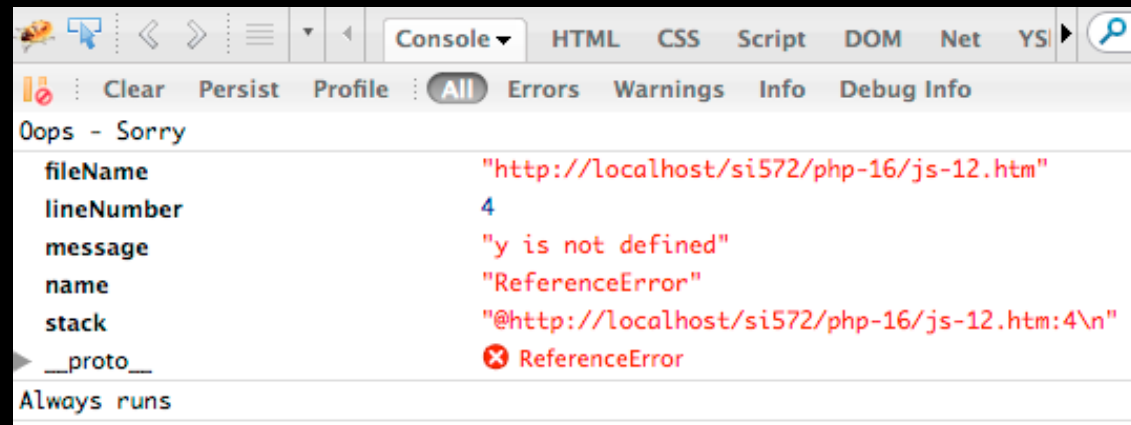
```
for (ball in balls) {  
    val = balls[ball];  
    if ( typeof val !== "string" ) continue;  
    console.log(ball+' = '+balls[ball]);  
}
```



Try / Catch

- Very similar to Java with a try / catch / finally pattern

```
try {  
    x = y + 1;  
    console.log(x);  
}  
catch(erro) {  
    console.log('Oops - Sorry');  
    console.dir(erro);  
}  
finally {  
    console.log('Always runs');  
}
```



Questions...