Free functions:

In JavaScript, functions like to run wild and free, belonging to the wilderness of the code, mating with each other like hippie binaries. These are cells. The building blocks of the language.

```
function iShitYouNot(aparameter) //notice no return type and no parameter
//types.
{
	var x = 1; //Variable declaration
	for(var i = 0; i < aparamenter; ++i)
	{
		x = x*(x+1);
	}

	return x; //Guess what this does?
}//Curly brackets FTW

function iPrint()
{
	console.log("Blah"); //System.out.println("Blah");
}
```

Free code:

These are bacteria. They don't belong to any organisms. They're just waiting to be parsed and executed. As soon as the js gets loaded, they will run when you hit them. As such there is no `main`, there is only free code waiting to be executed. There is one pitfall to parsing though:

```
ICallTheFuture(); //Causes parse ERROR

function ICallTheFuture()
{
    console.log("You can't call me yet, I haven't been parsed");
}
--------------------------------------------------------------------------
function ICallThePast()
{
    console.log("Good job");
}

ICallThePast();//This is legal. The following will be executed at load.
var x = 1;
for(var i = 0; i < aparamenter; ++i)
{
	x = x*(x+1);
}

console.log(x);
```

Funcject or Objection ?(Function == Object -> true;)

Objections are the organisms of JS. They are really just functions that are objects. Here's a standard build for a JS object.

```
function MyObj(param1, param2) //I am a constructor
{
    var privateVar = param1;
    this.publicVar = param2;

    //the keyword this behaves differently in JS than in Java. this refers to
    //the "owner" but the "owner" depends on the context. This is a story for
    //another day. self = this is a JS idiom to keep track of the owner of
    //the object

    var self = this;
    function privateFunction ()
    {
        private1 += 1;
    }

    this.privilegedIncrement = function ()
    {
        privateFunction();
    }

    this.priviledgedGet = function()
    {
        return privateVar;
    }
}
```

Java	JavaScript
Strongly-typed	Loosely-typed
Static	Dynamic
Classical	Prototypal
Classes	Functions
Constructors	Functions
Methods	Functions

There are a couple of things to note:

JavaScript is a **WEAKLY-typed** language == Variables don't have types. They are not the variables Gotham wants, they are variables Gotham needs.

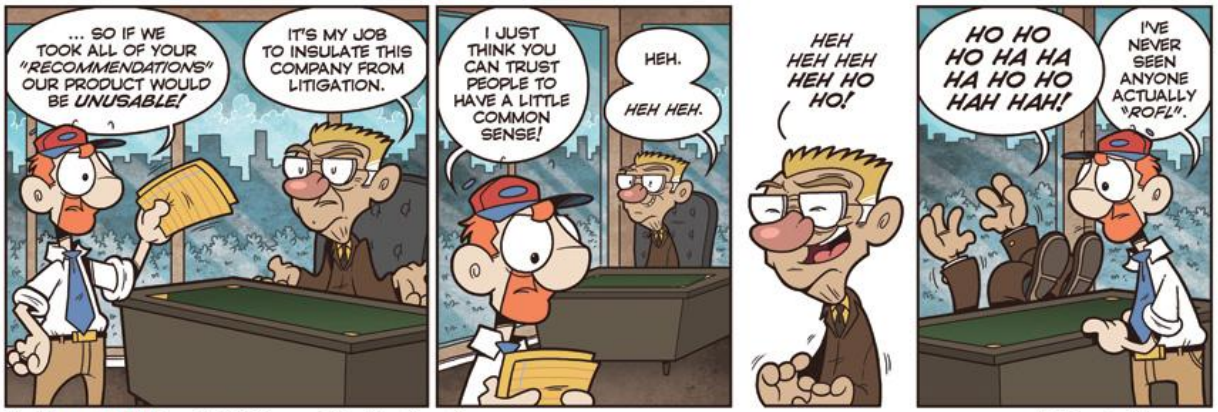
JavaScript is also **DYNAMICALLY-typed** but that's also a story for another day.

A JavaScript object is actually a function. It defines its constructor. You can see it two ways.
Every function is an object, or every object is a function. **Call them objections.**

There are 3 types of functions, and 2 types of data members.
Private data members are declared as local variables of the function with the keyword `var`, which are only accessed by *privileged* and *private* functions.
Public data members are declared as `this.publicVar`

Now, the real difference is *privileged* functions. They are declared inside the constructor and they are the only functions that can be accessed publicly, yet access private functions and variables. Private functions can, of course, access private data members.
Tl;dr *public* functions cannot access anything private. *privileged* functions can.

Using private members is not standard practice. Think Python. Trust your people to have common sense.



Arrays and Dictionaries

Here's an example on how to use Array's and Dictionaries

```
var blah = new Array();
blah[0]="1";  //1 element
blah[1]="2";  //2 elements
blah[2]= 3;   //3 elements
blah.push(4.01); // 4 elements
blah.push(function() {console.log("i have blahed");}); //5 elements
blah["Key"] = "Value"; //Not an element of the array

console.log(blah); //prints ["1", "2", 3, 4.01, function()]
blah[4].call(); //Prints "i have blahed"

if(blah.length == 5) blah.pop();
console.log(blah.length); //Prints 4
console.log(blah.toString());// Prints "1,2,3,4.01"
console.log(blah["Key"]);//Prints "Value"
```

As you can see, no types anywhere, *Array* contains a collection of different types, and everyone is happy. It even includes an objection (read function)! The objection here is an anonymous objection. They are essentially delegates. Things to note:

- *Array* implements *push* and *pop* operations
- All objects implement *toString()*.
- Element "Key" does not count to *length*. It can be used as a Dictionary ADT for KEY VALUE pairs. (If this means nothing to you, forget it)
- The anonymous objection can be called by using *call()*, since it doesn't have its own name.