# Programación orientada a objetos

con JavaScript

*Francisco Arce*
*www.pacoarce.com*

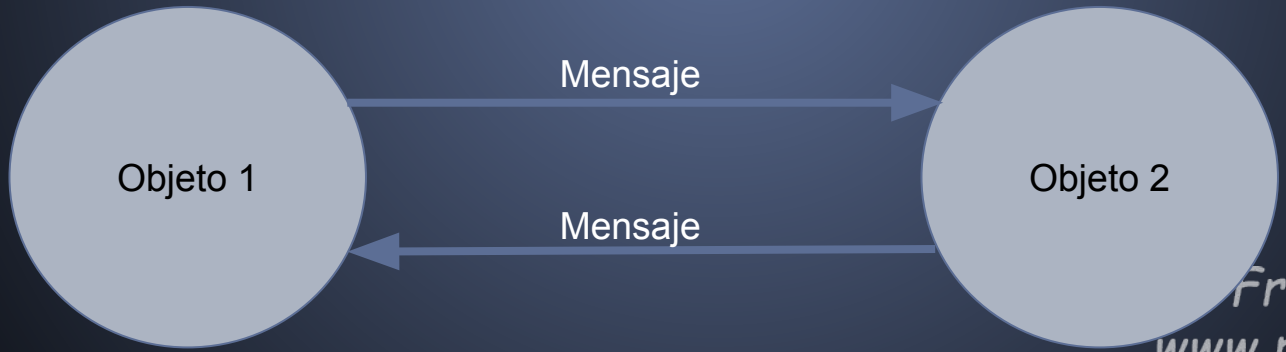
# Programación orientada a objetos

- Simula 67
- Smalltalk
- C++
- Java
- C#
- Objective-C
- JavaScript

Francisco Arce
www.pacoarce.com

# OOP con JavaScript

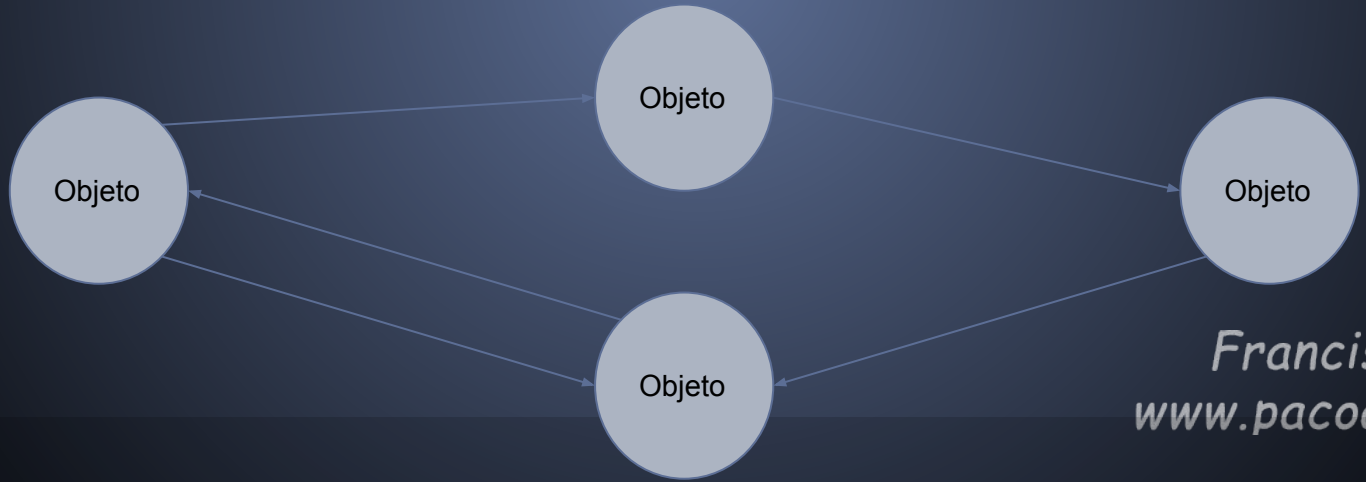
En la programación orientada a objetos, cada objeto es capaz de recibir mensajes, procesar datos y enviar mensajes a otros objetos.




Francisco Arce
www.pacoarce.com

# OOP con JavaScript

Cada objeto puede verse como una pequeña “máquina independiente” con un papel o responsabilidad definida.




Francisco Arce
www.pacoarce.com

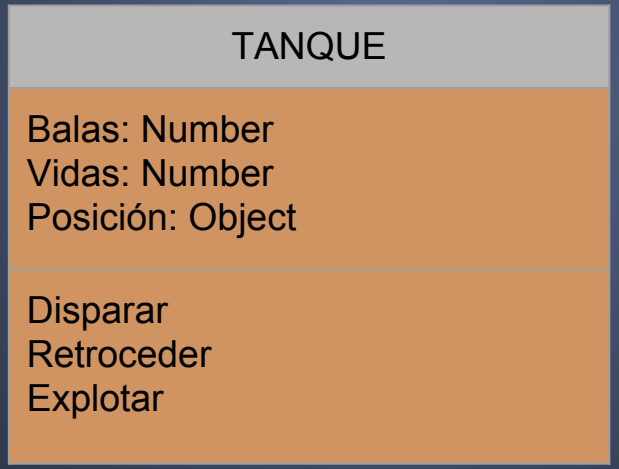
# Clases, objetos e instancias

OOP con JavaScript

Francisco Arce
www.pacoarce.com

# OOP con JavaScript

## Clase: Define las características del Objeto.




Francisco Arce
www.pacoarce.com

# OOP con JavaScript

## Objeto: Una instancia de una Clase.

| TANQUE |
| --- |
| Balas: Number<br>Vidas: Number<br>Posición: Object |
| Disparar<br>Retroceder<br>Explotar |

CLASE

INSTANCIAS

Francisco Arce
www.pacoarce.com

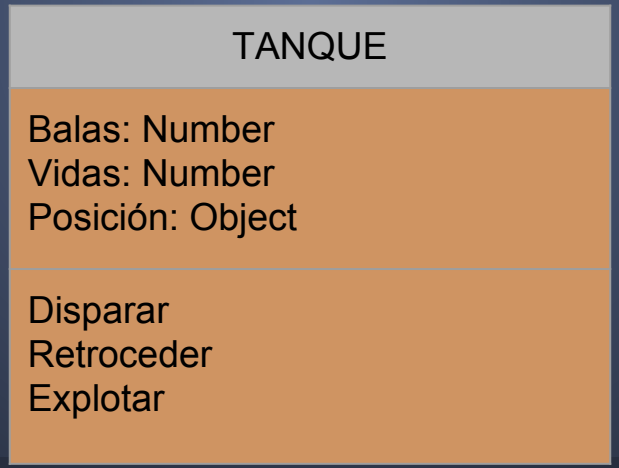
# Elementos de un clase

OOP con JavaScript

Francisco Arce
www.pacoarce.com

# OOP con JavaScript

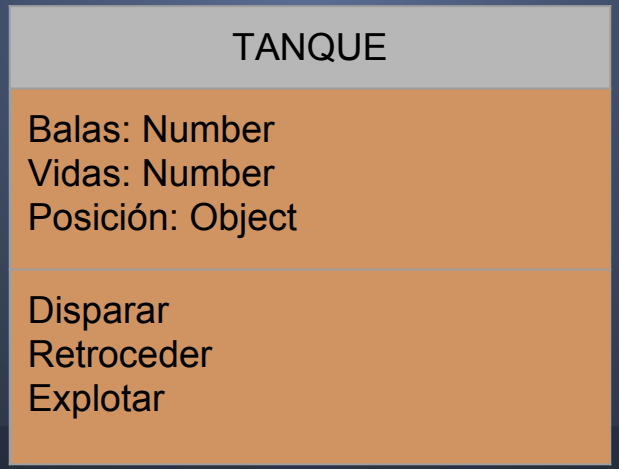
Propiedad: Una característica del Objeto, como el número de balas de un tanque.




Francisco Arce
www.pacoarce.com

# OOP con JavaScript

Método: Una capacidad del Objeto, como disparar().




Francisco Arce
www.pacoarce.com

# OOP con JavaScript

Constructor: Es un método llamado en el momento de la creación de instancias.

*Francisco Arce*
*www.pacoarce.com*

# Principios de la OOP

OOP con JavaScript

Francisco Arce
www.pacoarce.com

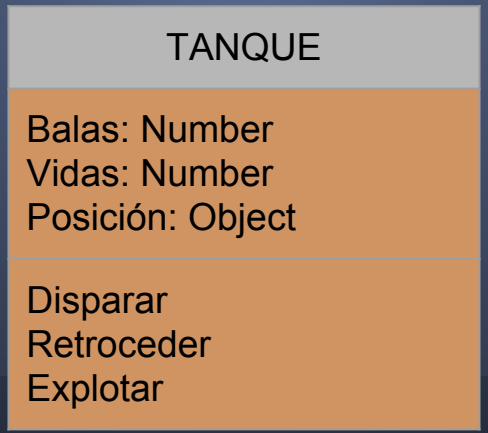
# OOP con JavaScript

Herencia: Una Clase puede heredar características de otra Clase.


Francisco Arce
www.pacoarce.com

# OOP con JavaScript

Encapsulamiento: Una Clase sólo define las características del Objeto, un Método sólo define cómo se ejecuta el Método.




Francisco Arce
www.pacoarce.com

# OOP con JavaScript

Abstracción: La conjunción de herencia compleja, métodos, propiedades que un objeto debe ser capaz de simular en un modelo de la realidad.

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

Polimorfismo: Diferentes Clases podrían definir el mismo método o propiedad.

| TANQUE |
| --- |
| Balas: Number<br>Vidas: Number<br>Posición: Object |
| Disparar<br>Retroceder<br>Explotar |

| AVION |
| --- |
| Balas: Number<br>Vidas: Number<br>Posición: Object |
| Disparar<br>Explotar<br>Despegar |

ancisco Arce
www.pacoarce.com

# La propiedad *prototype*

OOP con JavaScript

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

Las propiedades deben establecerse a la propiedad prototipo de la clase (función), para que la herencia funcione correctamente.

```
var tanque1 = new Tanque();
```

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

Para trabajar con propiedades dentro de la clase se utiliza la palabra reservada this , que se refiere al objeto actual.

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

El acceso (lectura o escritura) a una propiedad desde fuera de la clase se hace con la sintaxis:

```
tanque1.balas = 100;
```

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

Los ***métodos*** siguen la misma lógica que las propiedades, la diferencia es que son funciones y se definen como funciones.

```
instancia.metodo();
```

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

Llamar a un ***método*** es similar a acceder a una propiedad, pero se agrega () al final del nombre del método, posiblemente con argumentos.

```
tanque1.avanza(10,10);
```

Francisco Arce
www.pacoarce.com

# Programación orientada a prototipos

OOP con JavaScript

Francisco Arce
www.pacoarce.com

# OOP con JavaScript

La programación basada en ***prototipos*** es un estilo de programación orientada a objetos en la que las clases no están presentes y la reutilización de comportamiento (conocido como ***herencia*** en lenguajes basados en clases) se lleva a cabo a través de un proceso de decoración de objetos existentes que sirven de prototipos.

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

Este modelo también se conoce como programación sin clases, ***orientada a prototipos*** o basada en funciones.

*Francisco Arce*
*www.pacoarce.com*

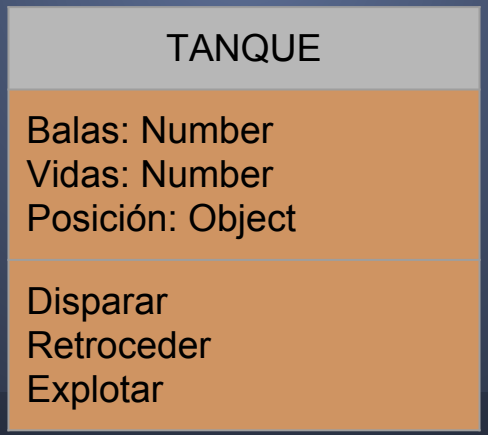
# UML

OOP con JavaScript

Francisco Arce
www.pacoarce.com

# UML

El lenguaje unificado de modelado o Unified Modeling Language (UML) permite visualizar las clases y las relaciones entre ellas.




Francisco Arce
www.pacoarce.com

# UML

En JavaScript, el constructor y el nombre de la clase son los mismos.

Luego se enumeran las propiedades y posteriormente los métodos.

# UML

Aunque en JS no se definen los tipos de dato, en el UML se indica el tipo de mismo.

Si el método no regresa ningún valor, se indica con :void.

# Links

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/class

https://en.wikipedia.org/wiki/ECMAScript

Francisco Arce
www.pacoarce.com

# Links

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Details_of_the_Object_Model

https://developer.mozilla.org/es/docs/Web/JavaScript/Introducción_a_JavaScript_orientado_a_objetos

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Funciones

Francisco Arce
www.pacoarce.com

# Programación orientada a objetos

con JavaScript

*Francisco Arce*
*www.pacoarce.com*

# La palabra reservada this

Con this indicamos que estamos trabajando “con este” objeto, sin necesidad de nombrarlo.

Francisco Arce
www.pacoarce.com

# Constructores

OOP con JavaScript

Francisco Arce
www.pacoarce.com

# Constructores

En JavaScript todo el código que se encuentre dentro de una función se ejecuta inmediatemente y se puede considerar un “constructor”. O sea, no hay constructor propiamente dicho.

Francisco Arce
www.pacoarce.com

# UML

OOP con JavaScript

*Francisco Arce*
*www.pacoarce.com*

# UML

El lenguaje unificado de modelado o Unified Modeling Language (UML) permite visualizar las clases y las relaciones entre ellas.

Francisco Arce
www.pacoarce.com

# UML

En JavaScript, el constructor y el nombre de l clase son los mismos.

Luego se enumeran las propiedades y posteriormente los métodos.

Francisco Arce
www.pacoarce.com

# UML

Aunque en JS no se definen los tipos de dato, en el UML se indica el tipo de mismo.

Si el método no regresa ningún valor, se indica con :void.

Francisco Arce
www.pacoarce.com

# Modificadores de acceso

OOP con JavaScript

*Francisco Arce*
*www.pacoarce.com*

# Modificadores de acceso

En JavaScript, en específico en ECMAScript 5 y anteriores, no hay modificadores de acceso, por lo que todas las propiedades y métodos son públicos.

Francisco Arce
www.pacoarce.com

# Modificadores de acceso

En JavaScript las funciones creadas dentro de la “clase” o mejor llamado métodos, son consideradas como otra variable.

Francisco Arce
www.pacoarce.com

# Referencias a clases externas

OOP con JavaScript

Francisco Arce
www.pacoarce.com

# Referencias externas

En JavaScript, podemos definir un método dentro de una “clase” y tener el código fuera de la misma, es decir, escribir el código en una función externa.

Francisco Arce
www.pacoarce.com

# Prototyping

OOP con JavaScript

Francisco Arce
www.pacoarce.com

# Prototyping

En JavaScript podemos añadir propiedades y métodos a una “clase” por medio del comando ***prototype***.

A la acción de añadir propiedades o métodos a una clase se le conoce como “prototyping”.

Francisco Arce
www.pacoarce.com

# Prototyping

Al usar ***prototype***, no se duplica el código en las diferentes instancias creadas, pero todos pueden utilizar estos recursos añadidos.

Francisco Arce
www.pacoarce.com

# Prototyping

En JavaScript, todas las funciones tienen un propiedad llamada ***prototype***, que es a su vez un objeto.

Francisco Arce
www.pacoarce.com

# Prototyping

Se puede utilizar prototype solo después de haber creado la función de la clase, no antes.

Francisco Arce
www.pacoarce.com

# Prototyping

Es una práctica común crear las propiedades dentro de la función de la clase, y añadir los métodos con ***prototype***.

Francisco Arce
www.pacoarce.com

# Prototyping

SI nosotros modificamos un método haciendo referencia a una instancia, sólo será modificado o sobreescrito (overwriting) esa instancia, y no todas las demás instancias.

# Propiedades y métodos estáticos

OOP con JavaScript

Francisco Arce
www.pacoarce.com

# Propiedades y métodos estáticos

Una propiedad estática no es otra cosa que una variable añadida a la función de la clase (sin prototype), pero por lo general se escriben con mayúsculas.

Francisco Arce
www.pacoarce.com

# Propiedades y métodos estáticos

Un método estático no es otra cosa que una función añadida a la función de la clase (sin prototype).

Francisco Arce
www.pacoarce.com

# Propiedades y métodos estáticos

En ambos casos pueden ser llamados sin necesidad de crear una instancia, sólo referenciarla con el nombre de la función clase.

Francisco Arce
www.pacoarce.com

# Propiedades privadas

OOP con JavaScript

*Francisco Arce*
*www.pacoarce.com*

# Propiedades privadas

Aunque JavaScript no tiene modificadores de acceso propiamente dichos como los demás lenguajes orientados a objetos, podemos hacer propiedades privadas si las definimos dentro de la función de clase con la palabra reservada var.

# Propiedades privadas

Es una buena práctica diferencias a estas propiedades de alguna manera. Por lo general se le antepone un guión bajo en el nombre.

Francisco Arce
www.pacoarce.com

# Espacio de nombres o name space

OOP con JavaScript

Francisco Arce
www.pacoarce.com

# OOP con JavaScript

Un espacio de nombres (name space) es un contenedor que permite asociar toda la funcionalidad de un determinado objeto con un nombre único.

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

En JavaScript un espacio de nombres es un objeto que permite asociarse a métodos, propiedades y objetos.

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

La idea de crear namespace en JavaScript es simple: Crear un único objeto global para las variables,  métodos y funciones, convirtiéndolos en propiedades de ese objeto.

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

El uso de los namespace permite minimizar el conflicto de nombres con otros objetos haciéndolos únicos dentro de nuestra aplicación.

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

JavaScript es un lenguaje basado en prototipos que no contiene ninguna declaración de clase, como se encuentra, por ejemplo, en C++ o Java.

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

En su lugar, JavaScript utiliza funciones como clases. Definir una clase es tan fácil como definir una función.

*Francisco Arce*
*www.pacoarce.com*

# OOP con JavaScript

Como una buena práctica, los nombres de espacio los escribiremos con mayúsculas.

Francisco Arce
www.pacoarce.com

# Programación orientada a objetos

Funciones

*Francisco Arce*
*www.pacoarce.com*

# ¿Qué es una función?

Francisco Arce
www.pacoarce.com

# ¿Qué es una función?

Una función en un conjunto de instrucciones que se llaman o invocan bajo un nombre y tienen un propósito definido y que puede regresar un valor.

Francisco Arce
www.pacoarce.com

# ¿Qué es una función?

```
function suma(a, b) {
   var c = a + b;
   return c;
}
```

Francisco Arce
www.pacoarce.com

# ¿Qué es una función?

- función (function)
- nombre
- propiedades
- instrucciones
- return

Francisco Arce
www.pacoarce.com

# ¿Qué es una función?

Una función tiene 2 parámetros:

- arguments
- prototype

Y tiene dos métodos:

- call()
- apply()

Francisco Arce
www.pacoarce.com

# ¿Qué es una función?

```
function sumaNumeros() {
  var i, tot = 0;
  var numeros = arguments.length;
  for (i = 0; i < numeros; i++) {
      tot += arguments[i];
  }
  return tot;
}
```

Francisco Arce
www.pacoarce.com

# Funciones predefinidas

*Francisco Arce*
*www.pacoarce.com*

# Funciones predefinidas

En JavaScript tenemos funciones predefinidas que no necesitan ninguna instancia para ser utilizadas:

- parseInt()
- parseFloat()
- isNaN()

Francisco Arce
www.pacoarce.com

# Funciones predefinidas

- isFinite()
- encodeURI()
- decodeURI()
- encodeURIComponent()
- decodeURIComponent()
- eval()

Francisco Arce
www.pacoarce.com

# Funciones anónimas

*Francisco Arce*
*www.pacoarce.com*

# Función anónima

No tiene nombre y por lo general es ejecutada en el momento. La podemos asignar como una variable.

Francisco Arce
www.pacoarce.com

# Función anónima

```
// asignamos la función a la variable saludo
 var saludo = function(hora)
{
  if (hora >= 22 || hora <= 5)
     document.write("Buenas noches");
  else
     document.write("Buenos días");
}
// llama a la función
saludo(10);
```

Francisco Arce
www.pacoarce.com

# Funciones de callback

*Francisco Arce*
*www.pacoarce.com*

# Función callback

Las funciones no son otra cosa que datos asignados a una variable, por lo que pueden ser copiados, borrados y llamados como parámetros.

Francisco Arce
www.pacoarce.com

# Funciones de callback

Una función puede ser pasada como un parámetro.

Francisco Arce
www.pacoarce.com

# Funciones que se autoinvocan

Francisco Arce
www.pacoarce.com

# Funciones que se auto invocan

Por medio de los paréntesis, podemos autoinvocar a una función, generalmente es una función anónima.

Podemos pasarle parámetros por medio de paréntesis.

Francisco Arce
www.pacoarce.com

# Funciones que se autoinvocan

```
(
   function(){
      alert('Hola, cara de bola');
   }
)()
```

Francisco Arce
www.pacoarce.com

# Funciones que se autoinvocan

```
(
   function(nombre){
      alert('Hola ' + nombre + '!');
  }
)('Crayola')
```

Francisco Arce
www.pacoarce.com

# Funciones dentro de funciones

Francisco Arce
www.pacoarce.com

# Funciones dentro de funciones

Se puede crear una función dentro de otra función, pero sólo será visible dentro de ésta.

Francisco Arce
www.pacoarce.com

# Funciones dentro de funciones

```
function a(numero) {
  function b(entrada) {
    return entrada * 2;
  };
  return 'Resultado ' + b(numero);
};
```

Francisco Arce
www.pacoarce.com

# Funciones dentro de funciones

```
var a = function(numero) {
   var b = function(entrada) {
      return entrada * 2;
   };
   return 'Resultado ' + b(numero);
};
```

Francisco Arce
www.pacoarce.com

# Funciones que regresan funciones

Francisco Arce
www.pacoarce.com

# Funciones que regresan funciones

Dentro de la sentencia return puedes ejecutar una función anónima.

Si una función no tiene una sentencia “return”, regresará una valor “undefined”.

Francisco Arce
www.pacoarce.com

# Funciones que regresan funciones

```
function a() {
  alert('Hola ');
  return function(){
            alert('cara de bola');
          };
}
```

Francisco Arce
www.pacoarce.com

# Redefinir una función

Francisco Arce
www.pacoarce.com

# Redefinir una función

Ya que una función puede regresar otra función, podemos utilizar la segunda función para redefinir a la primera.

Francisco Arce
www.pacoarce.com

# Redefinir una función

Ya que una función puede regresar otra función, podemos utilizar la segunda función para redefinir a la primera.

Francisco Arce
www.pacoarce.com

# call y apply

Francisco Arce
www.pacoarce.com

# call y apply

Ambas funciones sirven para llamar a otra función.

En ambas, el primer parámetro debe ser el objeto propietario.

Francisco Arce
www.pacoarce.com

# call y apply

En el método ***call***, los parámetros se pasan separados por comas.

En el método ***apply***, los parámetros se pasan como un arreglo.

Francisco Arce
www.pacoarce.com

# call y apply

```
function producto(a, b) {
   return a * b;
}
var objeto = producto.call(objeto, 7, 3);
alert(objeto);
```

Francisco Arce
www.pacoarce.com

# call y apply

```
function producto(a, b) {
   return a * b;
}
var objeto = producto.apply(objeto, [7, 3]);
alert(objeto);
```

Francisco Arce
www.pacoarce.com

# Programación orientada a objetos

con JavaScript

Francisco Arce
www.pacoarce.com

# ¿Qué es un objeto?

con JavaScript

Francisco Arce
www.pacoarce.com

# ¿Qué es un objeto?

En JavaScript “casi” todo es un objeto.

Los valores primitivos son: cadenas, números y booleanos.

Todos lo demás son objetos: arreglos, funciones, expresiones regulares y objetos.

Francisco Arce
www.pacoarce.com

# ¿Qué es un objeto?

Los objetos son variables que contienen variables, incluso arreglos, funciones y otros objetos.

Francisco Arce
www.pacoarce.com

# ¿Qué es un objeto?

Existen tres maneras de crear un objeto:

- Por medio de las llaves
- Por un constructor
- Por la palabra reservada “new”

# Elementos, propiedades y eventos de un objeto

con JavaScript

Francisco Arce
www.pacoarce.com

# Elementos, propiedades y métodos

En un arreglo se considera que tiene “elementos”.

En un objeto decimos que contiene “propiedades”.

Si un objeto contiene una función, se le considera un “método”.

Francisco Arce
www.pacoarce.com

# Elementos, propiedades y métodos

También podemos almacenar una función en un arreglo, pero es poco común.

Para acceder a una propiedad o método de un objeto, podemos utilizar:

- La notación punto (recomendada)
- Notación de corchetes (menos usada)

# Modificar propiedades y métodos de un objeto

con JavaScript

Francisco Arce
www.pacoarce.com

# Alterar métodos y propiedades

Podemos iniciar con un objeto vacío y agregarles métodos y propiedades en el tiempo de ejecución del script.

Francisco Arce
www.pacoarce.com

# Alterar métodos y propiedades

Borramos una propiedad con delete.

Añadimos por medio de la sintaxis punto.

Francisco Arce
www.pacoarce.com

# La palabra reservada “this”

con JavaScript

Francisco Arce
www.pacoarce.com

# La palabra reservada “this”

Cuando estamos dentro de un objeto, podemos hacer referencia a sus elementos (propiedades o métodos) por medio de la palabra reservada “this”.

Francisco Arce
www.pacoarce.com

# El constructor

con JavaScript

Francisco Arce
www.pacoarce.com

# El constructor

Podemos crear un objeto o instancia por medio del “constructor” por medio del operador “new”.

```
var tanque1 = new Tanque();
```

Francisco Arce
www.pacoarce.com

# El constructor

Por convención, la primera letra de un constructor o función de clase, va en mayúscula.

# Objetos globales

con JavaScript

Francisco Arce
www.pacoarce.com

# Objetos globales

Todos los elementos, propiedades o métodos, de un script, están colgados del objeto window y se consideran como globales.

# Objetos globales

Todos los elementos, propiedades o métodos, de un script, están colgados del objeto window y se consideran como globales.

# La propiedad construct

con JavaScript

Francisco Arce
www.pacoarce.com

# La propiedad constructor

Son las propiedades creadas dentro de la función constructora.

Francisco Arce
www.pacoarce.com

# El operador instaceof

con JavaScript

Francisco Arce
www.pacoarce.com

# El operador instanceof

Con el operador instanceof podemos saber el nombre del constructor o función de clase del objeto.

Francisco Arce
www.pacoarce.com

# Funciones que regresan objetos

con JavaScript

Francisco Arce
www.pacoarce.com

# Funciones que regresan objetos

Instead of returning the object this, which contains the property a, the constructor returned another object that contains the property b. This is possible only if the return value is an object. Otherwise, if you try to return anything that is not an object, the constructor will proceed with its usual behavior and return this.

Francisco Arce
www.pacoarce.com

# Pasar objetos a funciones

con JavaScript

Francisco Arce
www.pacoarce.com

# Pasar objetos en funciones

Los objetos se pasan a una función por referencia, las variables simples se pasan por valor.

Francisco Arce
www.pacoarce.com

# Comparación de objetos

con JavaScript

Francisco Arce
www.pacoarce.com

# Comparación de objetos

La comparación de objetos solo será verdadera si se comparan los objetos de un mismo origen.

Si los objetos son exactamente iguales, pero de diferente origen, el resultado será falso.

La igualdad estricta siempre da falso.

Francisco Arce
www.pacoarce.com

# Comparación de objetos

La comparación de objetos solo será verdadera si se comparan los objetos de un mismo origen.

Si los objetos son exactamente iguales, pero de diferente origen, el resultado será falso.

La igualdad estricta siempre da falso.

Francisco Arce
www.pacoarce.com

# Objetos precostruidos en JavaScript

con JavaScript

Francisco Arce
www.pacoarce.com

# Objetos preconstruidos en JS

Data wrapper objects—Object, Array, Function, Boolean, Number, and String.

Francisco Arce
www.pacoarce.com

# Objetos preconstruidos en JS

Utility objects—These are Math, Date, RegExp and can come in very handy.

Francisco Arce
www.pacoarce.com

# Objetos preconstruidos en JS

Error objects—The generic Error object as well as other, more specific objects that can help your program recover its working state when something unexpected happens.

Francisco Arce
www.pacoarce.com

# Objetos preconstruidos en JS

To create a new empty object you can use the literal notation or the Object() constructor function.

# Objetos preconstruidos en JS

Un objeto vacío contiene los siguientes elementos:

o.constructor

o.toString()

o.valueOf()

Francisco Arce
www.pacoarce.com

# Objetos preconstruidos en JS

Un arreglo o Array es otro objeto pre-construido.

Francisco Arce
www.pacoarce.com

# Objetos preconstruidos en JS

Una función es un objeto predefinido en JS.

Su constructor es Function()

Toda función tiene un elemento prototype que es a su vez un objeto.

Poseen un método llamado “call()” y “apply()”.

Una propiedad “arguments” y “callee”

Francisco Arce
www.pacoarce.com

# Objetos preconstruidos en JS

Los valores booleanos son otro Objeto preconstruido en JS

Francisco Arce
www.pacoarce.com

# Programación orientada a objetos

con JavaScript

*Francisco Arce*
*www.pacoarce.com*

# ¿Qué es el prototype?

en JavaScript

Francisco Arce
www.pacoarce.com

# ¿Qué es el prototype?

Las funciones son objetos que tienen propiedades (arguments, prototype y constructor) y métodos (call() y apply())

Francisco Arce
www.pacoarce.com

# ¿Qué es el prototype?

***prototype*** es una propiedad que tiene toda función y es de tipo objeto. En un inicio es un objeto vacío.

Francisco Arce
www.pacoarce.com

# ¿Qué es el prototype?

Por medio de la propiedad prototype podemos añadir métodos y propiedades a la función de clase o constructor.

Francisco Arce
www.pacoarce.com

# ¿Qué es el prototype?

El prototipo solo se utiliza cuando utilizas a la función como función constructora.

Francisco Arce
www.pacoarce.com

# Añadir propiedades al prototipo

JavaScript

Francisco Arce
www.pacoarce.com

# Añadir propiedades al prototype

Cuando creas una función como constructora por medio de “new”, puedes hacer referencia al mismo por medio de “this” (la cual no es una variable).

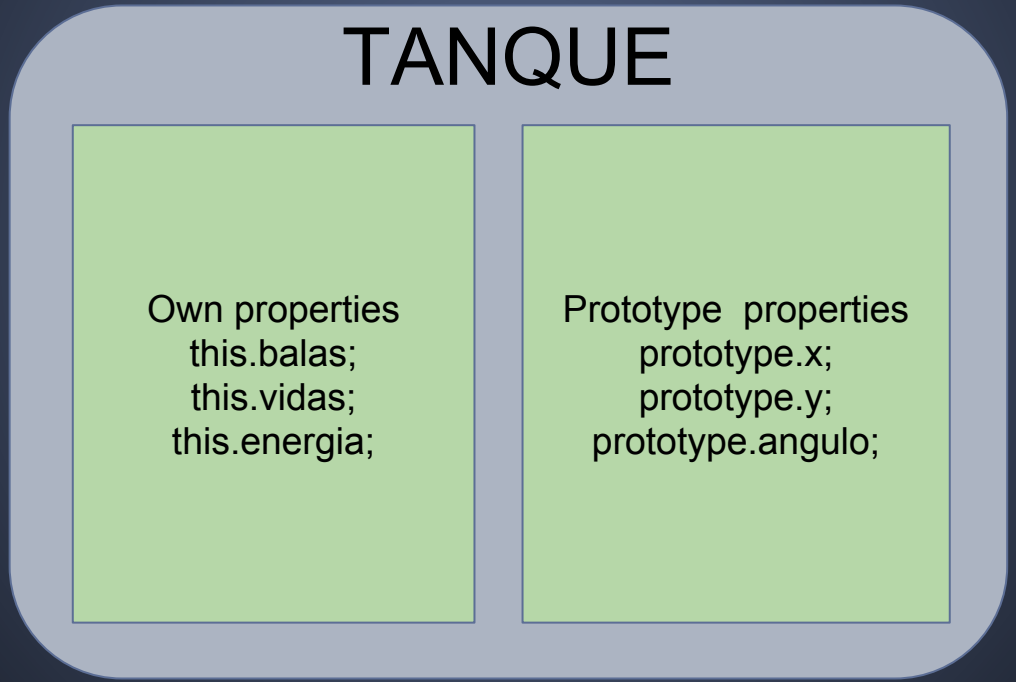
# Añadir propiedades al prototype

A las propiedades añadidas con “this” se les conoce como “own properties”.

A las propiedades añadidas con “prototype” se les conoce como “prototype properties”.

Francisco Arce
www.pacoarce.com

# Añadir propiedades al prototype




Francisco Arce
www.pacoarce.com

# Usar propiedades y métodos del prototipo

JavaScript

Francisco Arce
www.pacoarce.com

# Usar propiedades y métodos del prototipo

Como el prototipo es un objeto, cualquier modificación la pueden visualizar todas las instancias, incluso si fueron creadas antes de la modificación al prototipo.

Francisco Arce
www.pacoarce.com

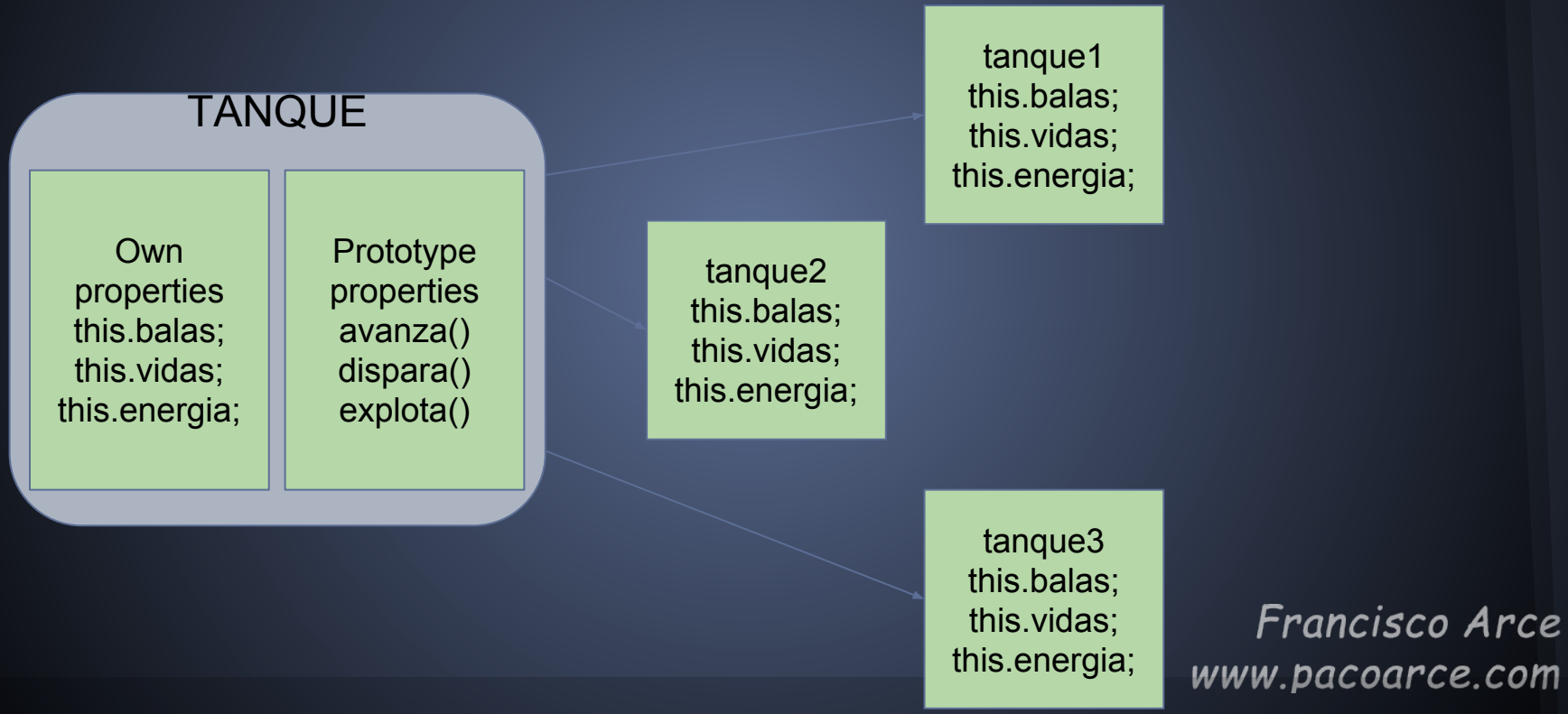
# Usar propiedades y métodos del prototipo

Todo lo que se encuentre en el prototipo se pasa “por referencia”, por lo tanto, no se duplica en cada instancia.

En cambio en las “own properties” si se duplican en cada instancia.

Francisco Arce
www.pacoarce.com

# Usar propiedades y métodos del prototipo




Francisco Arce
www.pacoarce.com

# Propiedades del objeto y propiedades del prototipo

JavaScript

Francisco Arce
www.pacoarce.com

# Prioridades del prototipo

JS primero busca en la función la propiedad, y si no existe la busca en el objeto prototype.

Si hay una propiedad de la función y una propiedad en el objeto prototype, la propiedad de la función tiene precedencia.

Francisco Arce
www.pacoarce.com

# Prioridades del prototipo

Si eliminas la propiedad de la función (own properties), aparecerá la propiedad prototype. A esto se le conoce como

"cadena de prototipos"

o

"property chain"

# Sobreescribir propiedades de la función

JavaScript

Francisco Arce
www.pacoarce.com

# Sobreescribir propiedades

Para que sepamos si la propiedad es de la función o del prototipo, tenemos el método hasOwnProperty().

Francisco Arce
www.pacoarce.com

# Sobreescribir propiedades

Si borramos una propiedad de la función, JavaScript nos regresará la propiedad del prototipo.

Francisco Arce
www.pacoarce.com

# Visualizar el contenido del prototipo

JavaScript

Francisco Arce
www.pacoarce.com

# Visualizar el contenido del prototipo

Por medio de un ciclo for...in podemos enlistar todas las propiedades de un objeto.

Por medio de l función hasOwnProperty() sabemos si son propiedades del objeto (true) o del objeto prototype (false).

Francisco Arce
www.pacoarce.com

# Visualizar el contenido del prototipo

Para saber si una variable es enumerable (visible en el for..in) utilizamos el método propertyIsEnumerable();

Francisco Arce
www.pacoarce.com

# isPrototypeOf()

JavaScript

Francisco Arce
www.pacoarce.com

# El método isPrototypeOf()

El método ***isPrototypeOf()*** nos indica si un objeto es prototipo de otro.

Regresa un valor booleano si son prototipos o un falso si no.

Francisco Arce
www.pacoarce.com

# __proto__

Francisco Arce
www.pacoarce.com

# Prototype

Todos los objetos en JavaScript tienen una propiedad prototype que es otro objeto.

En JavaScript, los objetos heredan por medio de la propiedad prototype.

El prototipo del objeto Object es el eslabón más alto en la cadena de prototipos.

Francisco Arce
www.pacoarce.com

# Prototype

Cualquier objeto hereda de Object.prototype.

La relación entre el objeto y la clase que le hereda es “__proto__”

No es buena idea usarla porque no es compatible (IE)

Francisco Arce
www.pacoarce.com

# Prototype

prototype pertenece a la función constructora

__proto__ pertenece a las instancias

Un sustituto de __proto__ es Object.
getPrototypeOf()

pero sólo corre en ES5, no en ES3.

Francisco Arce
www.pacoarce.com

# Prototype

Francisco Arce
www.pacoarce.com

# Modificar objetos preestablecidos

Francisco Arce
www.pacoarce.com

# Modificar los objetos predefinidos

Podemos añadir propiedades o métodos a los objetos propios de JS como Array, Object, Function, etc.

Douglas Crockford

cadena de prototipos o “prototype chaining”

# Links

https://www.packtpub.com/packtlib/book/Web%20Development/9781847194145/5

Francisco Arce
www.pacoarce.com