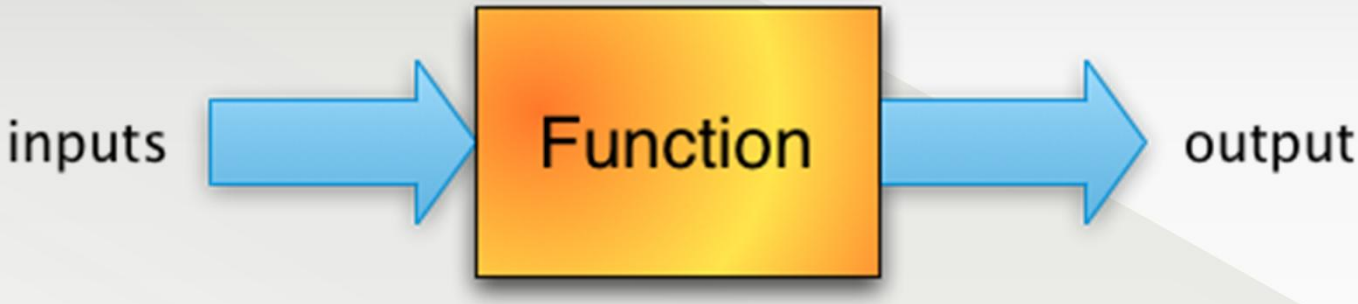
Definición de función

- Las funciones son uno de los elementos más importantes de cualquier lenguaje de programación actual, mucho más en JavaScript, que de hecho es presentado como un lenguaje funcional.
- Si bien es cierto, las funciones se pueden definir como un bloque de código independiente que se identifica con un nombre o identificador y que se puede invocar una o varias veces usando ese identificador para ejecutar las instrucciones definidas en la función. Además pueden recibir parámetros o argumentos y también devolver valores, cabe indicar que en JavaScript las funciones son objetos de primera clase.

Definición de función

- Que las funciones sean objetos de primera clase significa entre otras cosas que una función es un tipo especial de objeto que puede hacer lo que cualquier objeto puede hacer.



Definición de función

- Con la consideración que los objetos no han sido abordados en el curso por el momento, es preciso indicar desde ya todo lo que se puede hacer con las funciones en JavaScript por ser consideradas dentro del lenguaje como objetos de primera clase.
  - Una función es una instancia del objeto Object.
  - Las funciones pueden ser datos y por tanto asignarse a variables, a propiedades de un objeto, presentarse como elementos de una matriz, pasarse como argumentos de otras funciones o devolverse como valor de retorno de otra función.
  - Las funciones pueden ser métodos y asignarse como propiedad de un objeto.
  - Las funciones pueden ser constructores de un objeto.
  - Las funciones se pueden definir con el constructor Function().
  - Las funciones definidas como funciones inmediatas o funciones anónimas autoinvocadas.

Sintaxis de las funciones

```
function nombre_funcion([arg1, arg2,…, argn]){
     //sentencia(s);
}
```

Donde:

function, es una palabra reservada de JavaScript que le indica al navegador que lo que viene a continuación es una función.

nombre_funcion, es el nombre con el que se identificará a la función y mediante el cual se realizará la llamada.

arg1, arg2, … , argn, son los argumentos o parámetros que recibirá la función. Van entre corchetes porque son opcionales.

sentencia(s), representan las instrucciones que se ejecutarán al llamar a la función y con las cuales se realizará la tarea o cálculo para el que se ha diseñado la función.

Precaución con el uso de funciones

- Hay que poner especial atención a la hora de crear la funciones, para no repetir los nombres, principalmente porque esto no genera errores en JavaScript, y puede suponer quebraderos de cabeza cuando un *script* no funciona pero la consola de errores no muestra mensaje alguno.
- Si definimos dos funciones con el mismo nombre, como en este ejemplo:

```
function alerta(msg){
    alerta(msg);
}
function alerta(){
    alerta(msg);
}
```

- Sólo funcionará la segunda, que ha sido la última definida.

Llamada a una función

Desde un manejador de evento en una etiqueta HTML:

```
<etiqueta_html  manejador_evento=
"nombre_funcion([val1,val2,…,valN]);">
```

Desde un pseudo-enlace HTML:

```
<a href="javascript:nombre_funcion([val1,val2,…,valN]);">
```

o también, desde otra parte del script:

```
[variable=]nombre_funcion(valor1,valor2,valorN);
```

OBSERVACIÓN: Aunque esta forma de invocar a las funciones desde el código del documento HTML sigue siendo válida, es preciso señalar que las nuevas tendencias de codificación en donde se separa la funcionalidad proporcionada por JavaScript indican que ya no debe hacerlo así.

Manejadores de eventos

Las llamadas a las funciones pueden realizarse desde un manejador de eventos.

Un manejador de eventos es un mecanismo por el cual se pueden detectar las acciones que realiza el usuario sobre la página web.

Sintaxis de una llamada a función a través de un manejador de eventos:

```
<etiqueta_html nombre_manejador_evento="nombre_funcion([parametros]);">
```

Manejadores de eventos

- En el control avanzado de eventos definido en el DOM Nivel 2 (veremos en detalle el DOM y manejo de eventos en clases posteriores) permite utilizar funciones como controladores o manejadores de eventos.
- Estas funciones que trataremos como manejadores de eventos se activan al producirse ese evento en la página web.
- Mostremos un ejemplo de manejador de evento con el evento onload.

Manejadores de eventos

- Tenemos el siguiente documento:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Enlaces</title>
</head>
<body>
<h2>
  <a href="pagina.html" id="redirect">Google</a>
</h2>
</body>
</html>
```

Manejadores de eventos

- El código JavaScirpt:

```
window.onload = initAll;
function initAll(){
  document.getElementById("redirect").onclick = initRedirect;
}

function initRedirect(){
  window.location = "http://www.google.com";
  return false;
}
```

Manejadores de eventos de JavaScript

Atributo de evento	Descripción
onclick	Indica que se ha hecho clic en un elemento.
onfocus	Indica que el elemento ha recibido el foco; concretamente que se ha seleccionado para operar con él o para ingresar datos.
onload	Indica el evento de que una ventana o conjunto de marcos se han terminado de cargar.
onmouseover	Indica que el ratón se ha posicionado sobre el elemento.
onmouseout	Indica que el ratón se ha posicionado fuera del elemento.
onsubmit	Indica que el formulario va a ser enviado como producto de un clic sobre un botón de envío.
onreset	Indica que el formulario ha sido restablecido a sus valores iniciales como producto de la pulsación sobre un botón reset.

Paso de argumentos y valores devueltos

- Parámetros o argumentos
  - Las funciones pueden recibir parámetros o argumentos que funcionan como variables locales cuyo valor se establece en el momento en el que se invoca a la función desde alguna parte del ***script***.
  - Los valores que son enviados a una función para que realice o resuelva una tarea son denominados **argumentos** o **parámetros** de la función.
  - Casi siempre se necesitará que la función **realice una tarea**, para lo cual puede resultar conveniente **enviarle valores** a la función.

Paso de argumentos a las funciones

- Con respecto al paso de argumentos, debe tomar en cuenta que JavaScript trabaja con tipología débil de datos.
- Esto implica que los argumentos, al igual que ocurre con las variables, no se declaran de un tipo específico.
- Otro punto importante es el hecho que cuando se llama a una función con menos argumentos que los declarados, los argumentos no utilizados tendrán un valor *undefined*.

Devolución de valores

- Técnicamente, una función debería devolver un valor al punto del script desde donde se realiza la llamada a la función.
- JavaScript permite que se creen funciones que no devuelvan valor, pero realmente se está creando una función que devuelve un valor *undefined*.
- Para que la función devuelva un valor debe utilizarse la instrucción **return**.

Ejemplo de paso de argumentos

```
//Función que calcula la distancia entre dos puntos
//del plano cartesiano
function distance(x1, y1, x2, y2){
   var dx = x2 - x1;
   var dy = y2 - y1;
   return Math.sqrt(dx*dx + dy*dy);
}
-----------------------------------------------
//Función que calcula el factorial de un número
function factorial(x){
   if (x <= 1)
      return 1;
   return x * factoria(x-1);
}
```

Ámbito local y ámbito global

- Cuando se trabaja con funciones, es importante comprender el hecho que las variables que se declaran dentro del bloque de una función tienen ámbito local.
- En tanto que las variables que se definen afuera del bloque de una función tienen ámbito global.
- A las variables de ámbito local sólo se puede acceder desde dentro del bloque donde fueron definidas.

Ámbito local y ámbito global

- Las variables de ámbito global pueden ser accedidas desde dentro y desde fuera del bloque de una función.
- En el caso de existir una variable definida a nivel local y a nivel global, prevalecerá el valor definido en la variable local sobre la global dentro del bloque de una función, mientras que fuera de la función prevalecerá el valor definido a nivel global.

Formas de definir funciones en JavaScript

Forma tradicional con la instrucción ***function***	Como literales de función (**funciones anónimas**)	Funciones como métodos
Funciones constructoras	Usando el constructor Function()	Funciones inmediatas

Logotipo

Funciones con instrucción *function*

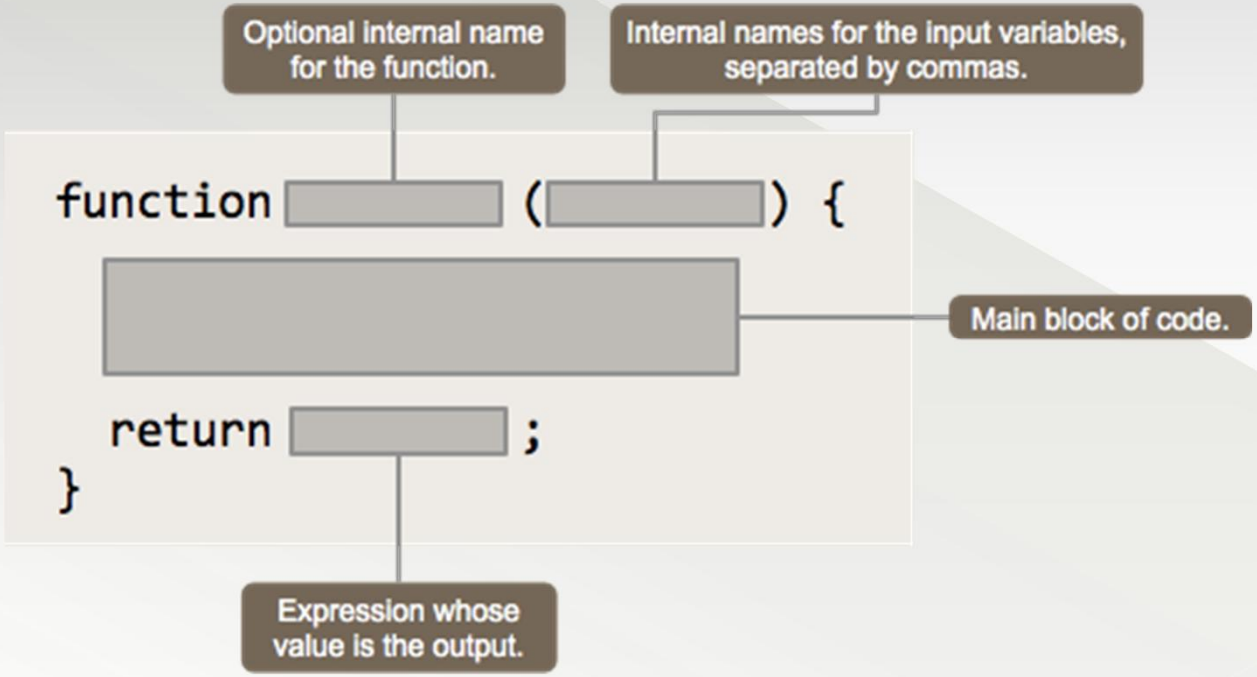
- El método más fácil de crear una función en JavaScript es utilizar la instrucción *function*.
- La sintaxis es la siguiente:

```
function functionName([arg1, arg2, …, argN]){
  //Instrucciones de la función;
  [return value;]
}
```

- En donde *function*, es la palabra clave que JavaScript utiliza para crear funciones.
- functionName representa el nombre o identificador de la función.
- [arg1, arg2, …, argN], representa el conjunto de argumentos que recibe la función. Los argumentos (cero, uno o más) son opcionales.
- [return value;] es el valor devuelto por la función. Tampoco es obligación devolver un valor.

Funciones con instrucción *function*

- En la siguiente imagen se ilustra lo que tradicionalmente se nos presenta como función, que también es válido utilizar en JavaScript.



Funciones con instrucción *function*

- Ejemplo 1:

```
function distance(x1,y1,x2,y2){
  var dx = x2 - x1;
  var dy = y2 - y1;
  return Math.sqrt(dx*dx + dy*dy);
}
----------------------------------------
function factorial(n){
  if(x<=1)
    return 1;
  return n * factorial(n-1);
}
```

Funciones con instrucción *function*

- Llamada o invocación a las funciones anteriores:

```
distance(3,1,5,6);
distance(2.3,5.2,1.2,4.7);
-----------------------------------------
factorial(0);
factorial(5);
factorial(parseInt(prompt("Ingrese el número","")));
factorial(parseInt(document.formName.txtnumber.value));
```

Funciones con literales de función

- A los literales de función también se les denomina funciones anónimas, por el hecho que no se declara en la instrucción un identificador o nombre para la función.
- Un literal de función es una expresión que define una función sin nombre.
- La sintaxis de un literal de función es prácticamente similar a la de una instrucción *function*, con la diferencia que no se utiliza como una instrucción si no como una expresión en la que no se requiere el nombre de la función.
- Ejemplos:

```
function f(x){ return x*x; }  //Usando instrucción function
var f = function(x){ return x*x; } //Usando literales de función
```

Funciones con literales de función

- Ejemplo:

```
function dividirNumeros(x, y){
  if(y==0) throw new Error("División entre cero");
  return x/y;
}
-------------------------------------------------------------
var dividirNumeros = function(x, y){
  if(y==0) throw new Error("División entre cero");
  return x/y;
}
-------------------------------------------------------------
//El código que invocará a la función dividir números
function init(){
  var n1 = document.form.number1.value;
  var n2 = document.form.number2.value;
  var calcular = document.getElementById("calcular");
  calcular.addEventListener("load", function(){
    dividirNumeros(n1/n2);
  }, false);
}
window.addEventListener("load", init, false);
```

Funciones como métodos

- Un método no es más que una función de JavaScript que se asigna en la propiedad de un objeto y se invoca a través de dicho objeto.
- Debe tener presente que en JavaScript las funciones son valores de datos y que no hay nada especial acerca de los nombres con los que se definen.
- Es por esa razón que una función se puede asignar a cualquier variable, o incluso, a la propiedad de un objeto.
- En el contexto de un método existe un objeto especial al que se le denomina *this*. Es el objeto mismo de la clase el que se invoca con esa palabra clave para hacer referencia a las propiedades y métodos definidos dentro de la clase.

Funciones como métodos

- Ejemplo:

```
//Definiendo la clase alumnoUDB haciendo uso de sintaxis de función
function alumnoUDB(nombre, apellido, edad, genero, carrera){
  //Propiedades de la clase
  this.nombre = nombre;
  this.apellido = apellido;
  this.edad = edad;
  this.genero = genero;
  this.carrera = carrera;
  this.numCarnet = null;
  //Métodos de la clase
  this.matricular = function(){
    var fecha = new Date();
    var year = fecha.getFullYear();
    var day = fecha.getDate();
    var sec = fecha.getSeconds();
    this.numCarnet = this.nombre.substring(0,1) + this.apellido.substring(0,1) +
this.formato(sec) + this.formato(day) + year;
  }
}
```

Funciones constructoras

- Básicamente una función o, más correctamente dicho en el lenguaje de la programación orientada a objetos, método constructor es el encargado de inicializar las propiedades definidas del objeto.
- Ejemplo:

```
function carro(manufacturer, model, year) {
  this.fabricante = manufacturer;
  this.modelo = model;
  this.año = year;
}
-------------------------------------------------
carronissan = new carro("Nissan", "Versa", 2007);
carrotoyota = new carro("Toyota", "Corolla", 2009);
carrohonda = new carro("Honda", "Civic", 2005);
```

Funciones con el constructor Function()

- JavaScript también permite crear funciones haciendo uso del constructor Function(). Recuerde que en JavaScript, todo o casi todo es objeto, incluso las funciones y por tanto, al ser objetos, las funciones también poseen un método constructor.
- Normalmente, crear una función con el constructor Function() es mucho más complejo que hacerlo con un literal de función o con la instrucción function, explicadas anteriormente. Por esta razón su uso no es muy común.

Funciones con el constructor Function()

- El constructor Function() puede recibir cualquier número de argumentos de cadena, cada uno de ellos son los parámetros que recibe la función a excepción del último que es el cuerpo o código que ejecutará la función.
- Si la función no recibirá argumentos, únicamente se proporciona un solo parámetros que sería el código que ejecutaría la función.
- El último argumento puede contener muchas instrucciones, no solo una. En el caso de tener más de una instrucción se separan por punto y coma cada una y encerrando todo el conjunto de instrucciones entre comillas.

Funciones con el constructor Function()

- Sintaxis:

```
var functionName = new Function("Arg1","Arg2",...,"Function Body");
```

- Ejemplo:

```
//Creando la función con el constructor Function()
var add = new Function("n1", "n2", "return n1+n2");
//Invocar la función anterior
var suma = add(2.75, 11.5);
```

Funciones inmediatas

- Las funciones inmediatas, denominadas también funciones anónimas autoinvocadas, representan una construcción importante del lenguaje JavaScript que se basa en el correcto uso de las clausuras(1).
- La sintaxis básica de una función inmediata o autoinvocada es la siguiente:

```
(function(){ //Instrucciones; })()
```

NOTAS: (1)Una clausura es el ámbito que se crea cuando se declara una función permitiendo que dicha función acceda y manipule variables externas a ella. En otras palabras, una clausura hace que una función acceda a todas las variables y a otras funciones que están en el ámbito cuando se declara.

Funciones inmediatas

- La construcción anterior posee tres partes que son:
  1. La función anónima.
     - `(function(){})`
  2. El código de la función en sí.
     - `//Instrucciones`
  3. Los paréntesis añadidos al final de todo
     - `()`
- Los paréntesis del final hacen que lo que está antes sea ejecutado de forma inmediata.
- Ya sabemos que una función se invoca usando la siguiente sintaxis: nombreFuncion(), pero como en JavaScript podemos usar cualquier expresión que se refiera a una de sus instancias.

Funciones inmediatas

- Por ejemplo:

```
var someFunction = function(){ … }
result = someFunction();
```

- Lo siguiente sería absolutamente equivalente y válido:

```
var someFunction = function(){ … }
result = (someFunction)();
```

- La diferencia es que el primer juego de paréntesis se utiliza como un mero delimitador, como cuando agrupamos una expresión matemática como: (x+5)*3, mientras que el segundo juego de paréntesis se interpreta como un operador.
- Al final, todo lo que está dentro del primer juego de paréntesis es considerado una referencia a la función que se va a ejecutar.

Funciones inmediatas

- Ejemplo:

```
(function(){
  var numclicks = 0;
  document.addEventListener("click", function(){
    alert("Van: " + (++numclicks) + " clicks");
  }, false);
})();
```