Básico

O jQuery é um framework construído usando recursos *JavaScript*. Assim, é possível usar todas as funções e outros recursos disponíveis em *JavaScript*. Esta aula explicaria a maioria dos conceitos básicos, mas frequentemente usado no jQuery.

String

Uma string em *JavaScript* é um objeto imutável que contém nenhum, um ou muitos caracteres. A seguir estão os exemplos válidos de uma string JavaScript;

```
"This is JavaScript String"
'This is JavaScript String'
'This is "really" a JavaScript String'
"This is 'really' a JavaScript String"
```

Números

Os números em JavaScript são valores que seguem o padrão IEEE 754 de 64 bits e precisão dupla. Eles são imutáveis, assim como as strings. A seguir estão os exemplos válidos de um número JavaScript;

```
5350
120.27
0.26
```

Boolean

Um booleano em JavaScript pode ser verdadeiro ou falso. Se um número for zero, o padrão será false. Uma string vazia por padrão é false.

A seguir estão os exemplos válidos de um Booleano JavaScript;

```
true      // true
false     // false
0         // false
1         // true
""        // false
"hello"   // true
```

Objetos

JavaScript suporta o conceito de Objeto muito bem. Pode criar um objeto da seguinte forma:

```
var emp = {
   name: "Zara",
   age: 10
};
```

Pode escrever e ler propriedades de um objeto usando a notação de ponto:

```
// Getting object properties
emp.name  // ==> Zara
emp.age   // ==> 10

// Setting object properties
emp.name = "Daisy"  // <== Daisy
emp.age  =  20      // <== 20
```

Arrays

Pode definir *arrays* usando o literal da matriz da seguinte forma:

```
var x = [];
var y = [1, 2, 3, 4, 5];
```

Uma *array* tem uma propriedade **length** que é útil para iteração:

```
var x = [1, 2, 3, 4, 5];

for (var i = 0; i < x.length; i++) {
   // Do something with x[i]
}
```

Funções

Uma função em JavaScript pode ser nomeada ou anônima. Uma função nomeada pode ser definida usando a palavra-chave da função da seguinte forma:

```
function named(){
   // do some stuff here
}
```

Uma função anônima pode ser definida de maneira similar a uma função normal, mas não teria nenhum nome.

Uma função anônima pode ser atribuída a uma variável ou passada para um método como mostrado abaixo:

```
var handler = function (){
   // do some stuff here
}
```

JQuery faz um uso de funções anônimas com muita freqüência como segue:

```
$(document).ready(function(){
   // do some stuff here
});
```

Argumentos

As variáveis de argumento do JavaScript são um tipo de array que possui a propriedade **length**. O exemplo a seguir explica:

```
function func(x){
   console.log(typeof x, arguments.length);
}

func();                 //==> "undefined", 0
func(1);                //==> "number", 1
func("1", "2", "3");    //==> "string", 3
```

Um objeto do argumento também possui uma propriedade **callee**, que se refere à função da qual você está dentro. Por exemplo:

```
function func() {
   return arguments.callee;
}

func();                 // ==> func
```

Contexto

No JavaScript a famosa palavra-chave **this** sempre se refere ao contexto atual. Dentro de uma função, este contexto pode mudar, dependendo de como a função é chamada:

```
$(document).ready(function() {
   // this refers to window.document
});

$("div").click(function() {
   // this refers to a div DOM element
});
```

Você pode especificar o contexto para uma chamada de função usando os métodos de **call()** e **apply()**.

A diferença entre eles é como eles passam argumentos. Call passa todos os argumentos como argumentos para a função, enquanto apply aceita um array como os argumentos.

```
function scope() {
   console.log(this, arguments.length);
}

scope() // window, 0
scope.call("foobar", [1,2]);  //==> "foobar", 2
scope.apply("foobar", [1,2]); //==> "foobar", 1
```

Escopo

O escopo de uma variável é a região do seu programa na qual ela é definida. A variável JavaScript terá apenas dois escopos.

- **Variáveis globais** - Uma variável global tem escopo global, o que significa que ela é definida em qualquer lugar no seu código JavaScript.
- **Variáveis locais** - Uma variável local será visível apenas dentro de uma função onde é definida. Os parâmetros de função são sempre locais para essa função.

Dentro do corpo de uma função, uma variável local tem precedência sobre uma variável global com o mesmo nome:

```
var myVar = "global";     // ==> Declare a global variable

function ( ) {
   var myVar = "local";   // ==> Declare a local variable
   document.write(myVar); // ==> local
}
```

Callback

Um Callback é uma função JavaScript simples passada para algum método como um argumento ou opção. Alguns callbacks são apenas eventos, chamados para dar ao usuário a chance de reagir quando um determinado estado é acionado.

O sistema de eventos do jQuery usa esses callbacks em todos os lugares, por exemplo:

```
$("body").click(function(event) {
   console.log("clicked: " + event.target);
});
```

A maioria dos callback fornece argumentos e um contexto. No exemplo do manipulador de eventos, o retorno de chamada é chamado com um argumento, um evento.

Alguns callbacks são necessários para retornar algo, outros tornam esse valor de retorno opcional. Para evitar um envio de formulário, um manipulador de eventos de envio pode retornar false da seguinte forma:

```
$("#myform").submit(function() {
   return false;
});
```

Closures

Os Closures são criados sempre que uma variável definida fora do escopo atual é acessada de dentro de algum escopo interno.

O exemplo a seguir mostra como a variável **counter** é visível nas funções de criação, incremento e impressão, mas não fora delas.

```
function create() {
   var counter = 0;

   return {
      increment: function() {
         counter++;
      },
           print: function() {
         console.log(counter);
      }
   }
}

var c = create();
c.increment();
c.print();     // ==> 1
```

Esse padrão permite criar objetos com métodos que operam em dados que não são visíveis para o mundo externo. Deve-se notar que a ocultação de dados é a base da programação orientada a objetos.

Funções Implementadas

JavaScript vem junto com um conjunto útil de funções internas. Esses métodos podem ser usados para manipular Strings, Números e Dados.

A seguir estão as funções importantes do JavaScript:

Sr.No.	Método e Descrição
1	**charAt()** Retorna o caractere no índice especificado.
2	**concat()** Combina o texto de duas strings e retorna uma nova string.
3	**forEach()** Chama uma função para cada elemento no array.
4	**indexOf()** Retorna o índice dentro do objeto String de chamada da primeira ocorrência do valor especificado, ou -1 se não for encontrado.
5	**length()** Retorna o comprimento da string.
6	**pop()** Remove o último elemento de um array e retorna esse elemento.
7	**push()** Adiciona um ou mais elementos ao final de um array e retorna o novo comprimento do array.
8	**reverse()** Inverte a ordem dos elementos de um array - o primeiro torna-se o último e o último torna-se o primeiro.
9	**sort()** Classifica os elementos de um array.
10	**substr()** Retorna os caracteres em uma string começando no local especificado através do número especificado de caracteres.
11	**toLowerCase()** Retorna o valor da string convertendo em minúscula.
12	**toString()** Retorna a representação da string em um valor do número.
13	**toUpperCase()** Retorna o valor da string convertendo em maiúsculas.

Figure 1: Métodos JavaScript