Multimedia im Netz

Online Multimedia

Winter semester 2015/16

Tutorial 06 – Major Subject

Today's Agenda

- Quick Test
- Advanced JavaScript and jQuery
- Break-Out Code-A-Long: AJAX Spotify Search
- Quiz

Quick Test

(10 Minutes)

Advanced JavaScript

Nested Functions: Scopes

```
function nest(){
    function getOne(){
        return 1;
    }
    function getTwo(){
        return 2;
    }
    console.log(getOne() + getTwo());
}
nest(); // 3
console.log(getOne() + getTwo()); // undefined!
```

Const, var, let: Mind the difference

- **`const`**: immutable constant
  - Performance benefits through compilation (modern JS engines compile)
  - Warnings if developer tries to override the value
- **`var`**: mutable variable scoped to nearest **function**

```
var y = 'five';
var y = 5;
console.log(y); // 5
```

- **`let`**: mutable variable scoped to nearest **block**

```
'use strict';
function letDemo(){
    var one = 0;
    for(let two = 0; two<10;two++){
        var three = two;
        console.log(one);
    }
    console.log(three); // 9
    console.log(two); // undefined;
}
```

Object oriented JavaScript (I)

- When programming with JavaScript you encounter various issues at some point:
  - Irresponsible usage of global variables
  - Variables are overridden unintentionally
  - Conflicts when including multiple JavaScript-files
  - Loss of readability
  - … much more…

- JavaScript fully supports object oriented programming to help you cope with some of these issues.

Object oriented JavaScript (II)

- There are different options to create objects in JavaScript:
  - Constructor functions (~ prototypes)
  - Object literal notation

- Which option should you prefer?
  - ... it depends on the problem at hand....
  - Constructors:
    - Useful if you need multiple instances of an object
  - Object literal notation:
    - If you only need one instance of an object
    - Useful for namespacing.

Constructed Objects

```
function Rabbit(){
    this.adjective = 'fat';
    this.whatAmI = function(){
        console.log('I am a ' + this.adjective + ' Rabbit!');
    };
    console.log(this.adjective + ' rabbit is alive');
}

var rabbit = new Rabbit();
rabbit.whatAmI();
rabbit.adjective = 'cool';
rabbit.whatAmI();
```

- Using **this** allows you to create member variables and methods
- You need to instantiate the object with the **new** keyword
- Not using the new keyword is similar to a regular function call, but you cannot access any properties later on.

Constructor with parameters

```
function Dog(adjective){
    this.adjective = adjective;
    this.whatAmI = function(){
        console.log('I am a ' + this.adjective + ' Dog!');
    }
}
var dooge = new Dog('hot');
dooge.whatAmI();

var wuff = new Dog('loud');
wuff.whatAmI();
```

Object Literal Notation (similar to JSON)

```
var horse = {
    adjective : 'crazy',
    whatAmI : function(){
        console.log("I am a " + this.adjective + " Horse!");
    }
};

var horse2 = horse; // this is just a reference, not a copy!

horse.whatAmI();    // crazy horse

horse.adjective = 'super weird';

horse2.whatAmI();   // horse2 is now also 'super weird'!
```

Accessing Object Attributes

```
var myObj = {};
var obj = new Object();
var str = "myString";
var rand = Math.random();

myObj.type = "Dot syntax";
myObj["date created"] = "String with space";
myObj[str] = "String value";
myObj[rand] = "Random Number";
myObj[obj] = "Object";
myObj[""] = "Even an empty string";
```

Advanced jQuery

Method Chaining

- Basically, any jQuery method returns another jQuery object, that you can now work with.
- (Possible) Advantages: readability, re-use of selection results
- Examples:

```
$("#myDiv").removeClass("off").addClass("on");
```

```
$("#myDiv")
        .css("color", "#cccccc")
        .removeClass("container")
        .attr("id", "someNewID")
        .append("<span>Text</span>");
```

DOM-Traversal

- Traverse the DOM tree with CSS selectors and jQuery methods
- Useful to select and edit elements efficiently.

- Examples:
  - `$("#myDiv").next("div")`
    gets the first `div`-element that follows the elemtent with ID `myDiv`
  - `$("ul").find("li.item")`
    finds all `li`–elements having the class `item` withing all `ul`-elements
  - More Examples: http://api.jquery.com/category/Traversing/

$.each()

- Convenient „for-loop shortcut“

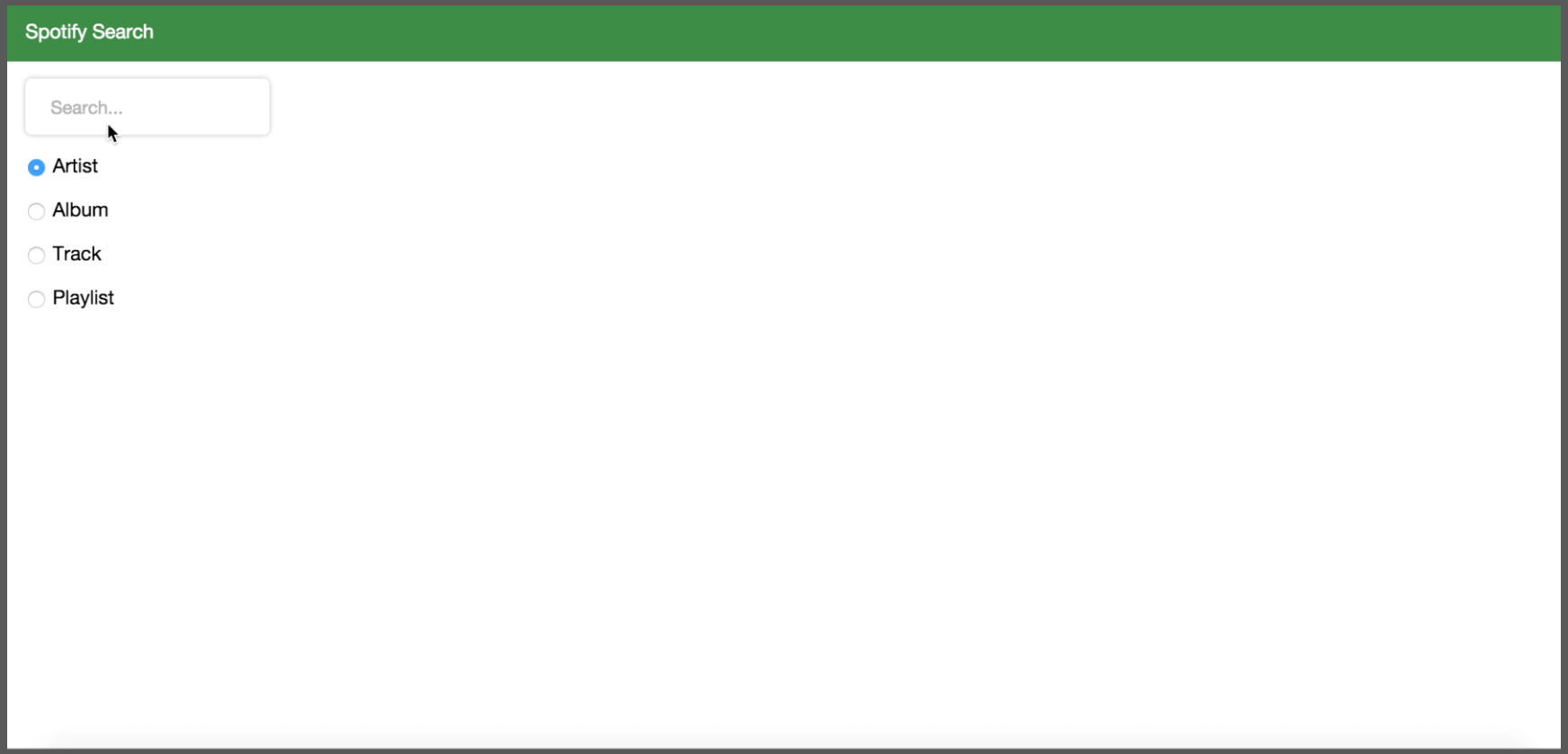
```
$.each(collection,function(index,item){
    // ...
});

// is equivalent to:

for(let i=0;i<collection.size();i++){
    let item = collection.get(i);
}
```

$(document).ready()

- `$(document).ready(onLoadCallback)` ensures that the DOM has been loaded entirely and that you can access any element inside it.
- Once the DOM is loaded, the callback is fired

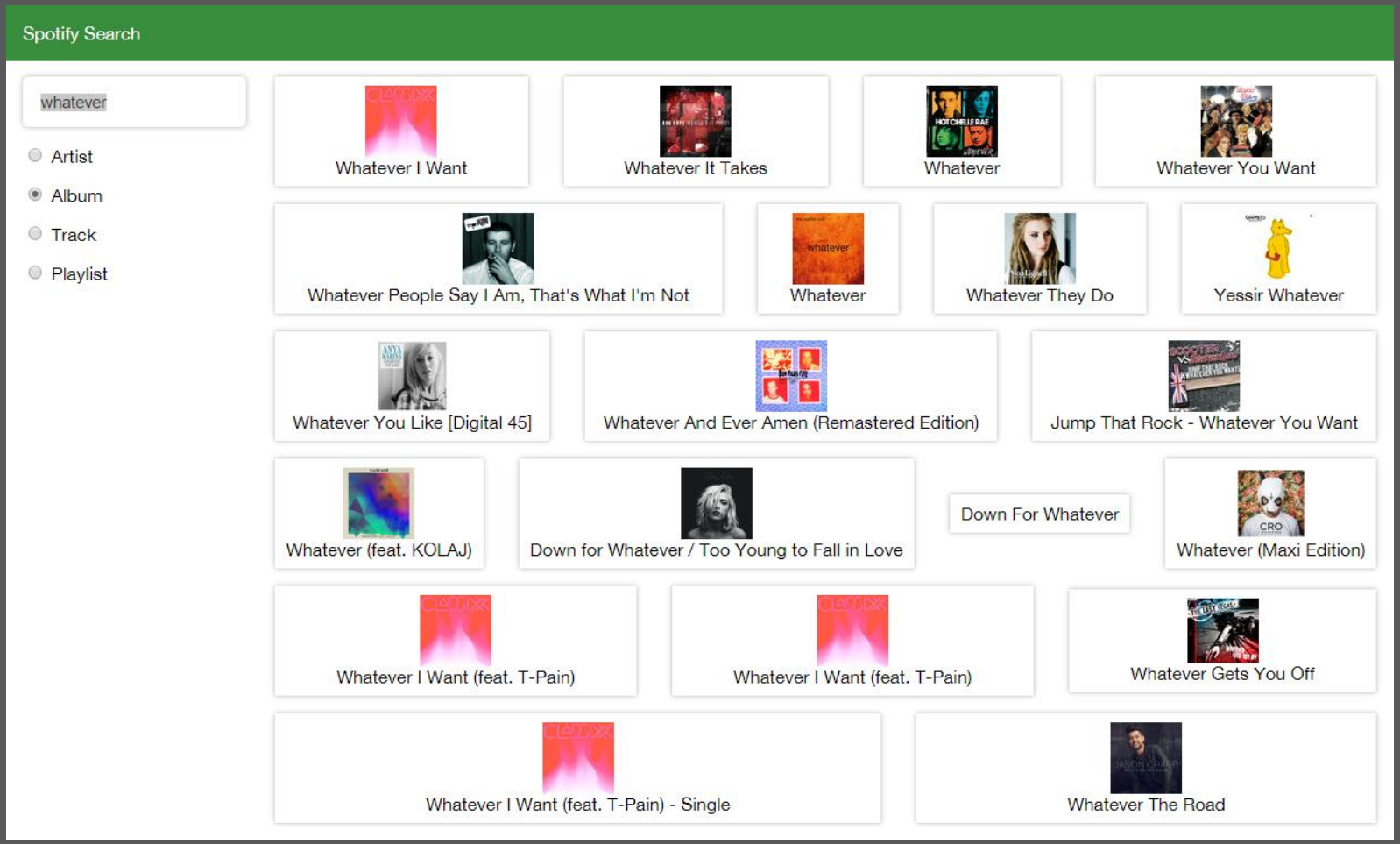
```
$(document).ready(function(){
    //JavaScript-Code
});
```

Breakout Code-A-Long



https://youtu.be/rJaZQ7tjLp0

Breakout Code-A-Long



Round-up Quiz

1. Why do we need the “new” keyword?
2. What’s the output of this:

```
var obj = {
    prop : 1
};
obj.prop ++;
var tst = obj;
console.log(tst.prop); // ?
```

3. Name an advantage of method chaining.
4. Why do we often need to put our code into the handler of `$(document).ready(handler)`?

Thanks!

What are your questions?

Let’s begin with the Assignment!

- Download the assignment sheet
- Start with task 1
- You can collaborate with your neighbor

- Turn in the assignment by **Wednesday** December 2nd, 12:00 noon via UniWorX