Restricted Deep Copy Implementations

This lesson will teach two methods to implement deep copy and their limitations.

JSON methods

There is a very easy implementation for making deep copies of JavaScript objects. Convert the JavaScript object into a JSON string, then convert it back into a JavaScript object.

```
var deepCopy = function( o ) {
    return JSON.parse(JSON.stringify( o ));
}
```

Restrictions:

- Object `o` has to be finite, otherwise `JSON.stringify` throws an error.
- The `JSON.stringify` conversion has to be lossless. Therefore, methods are not allowed as members of type `function` are ignored by the JSON stringifier. The `undefined` value is not allowed either. Object keys with `undefined` value are omitted, while undefined values in arrays are substituted by `null`.
- Language hacks won't work. Associative properties of an array will not appear in the cloned object. Example: `a = []; a.b = language hack';`. The value `a.b` will be accessible in the original object, but it will disappear from the deep copy.
- The prototype of the copy becomes `Object`. All properties coming from the prototype chain will be discarded.
- Members of the Date object become ISO-8601 strings, not representing the timezone of the client. The value `new Date(2011,0,1)` becomes `"2010-12-31T23:00:00.000Z"` after executing the above deep copy method in case you live in Central Europe.

I have hardly ever found any problems with these restrictions when using JSON methods to implement the deep copy functionality. It is still important to know what can potentially go wrong.

jQuery Extend

```
 var deepCopy = function( o ) {
    return $.extend( true, {}, o );
}

var shallowCopy = function( o ) {
    return $.extend( {}, o );
}
```

When the first argument of `$.extend` is `true`, it performs a deep extension. Deeply extending the empty object is the same as cloning the object.

Restrictions:

- Object o has to be finite, otherwise `$.extend` throws a `RangeError` for exceeding the maximum stack size
- The prototype of the copy becomes `Object`. All properties coming from the prototype chain will be added as own properties

There are multiple implementations for performing a deep clone. Most of them are restricted in some way. Relying on a generic deep cloning method is hardly ever needed. Whenever an object becomes too complex to clone, it is a good sign that the code has to be refactored.

Know what you're cloning

Cloning in JavaScript is not straightforward. As there is no native implementation for cloning, we have to implement it ourselves or we have to rely on a third party solution. These solutions work perfectly under some restrictions. Be aware of whether you perform deep or shallow cloning and also consider the restrictions. Whenever simple cloning solutions don't work, more often than not, think about refactoring your code instead of looking for a more reliable cloning function.