Exercise 2 - More JavaScript and HTML5

Søren Krogh - 20105661
Emil Madsen - 20105376
K. Rohde Fischer - 20052356

November 18, 2014

Embedded JavaScript

```
1  void(window.setInterval(function() {document.title = new Date}, 1000))
```

Running this code in the browser, returns 'undefined', and changes the title of the page to the date and time. the time is updated every second. In JavaScript the void operator allows expressions that produce side effects into places where an expression that evaluates to 'undifined' is desired. Hence 'undefined' is returned.

```
1  foo = { bar: "CWP" }
2  bar = "bar"
3  foo.undefined = foo[bar]
4  foo[foo.bar.baz]
```

If this pice of code is run, "CWP" is returned. First a new variable 'foo' is declared and a new object 'bar' with value "CWP" is constructed. Next 'bar' is set to be equal to the string "bar". Then 'foo.undefined' that otherwise would return 'undefined' is set to be equal to 'foo[bar]' which returns "CWP". Because 'foo.undefined' is set to 'foo[bar]'executing 'foo[foo.bar.baz]' returns "CWP".

```
1  function Msg(m) {this.msg = m; this.number = ++Msg.prototype.counter;}
2  Msg.prototype.counter = 0;
3  Msg.prototype.show = function() {alert("Message number " + this.number + " is: " + this.
4  var x = new Msg("hello");
5  var y = new Msg("world");
6  x.show();
7  y.show();
8  x.counter;
```

This will show two dialog boxes, the first saying "Message number 1 is:hello" and the second "Message number 2 is:world" and then return 2. First the function 'Msg' which takes one parameter is created. The function sets 'this.msg' to the parameter, and sets 'this.number' to ++"the function counter". Next the

counter is set to 0. Then the property 'show' is added to the 'Msg' objects. 'show' creates an alert message using 'this.number' to get the number of the message and 'this.msg' to get the actual message. The two messages 'hello' and 'world' is created and assigned to the two variables 'x' and 'y' respectively. Then the two messages are shown using the 'show()' property. Lastly 'x.counter' is executed which returnes the current state of the Msg counter, which at this state afte creating the two messages is 2 .

```
function Person(n) {
  this.name = n || "???"
  Person.prototype.count++
}
Person.prototype.count = 0
function Student(n,s) {
  this.base = Person
  this.base(n)
  delete this.base
  this.studentid = s
}
Student.prototype = new Person
var x = new Student("Joe Average", "100026")
x.count
```

Runnig this will return the value 2. First the function 'Person' is created. This takes one parameter and creates a Person object with the given parameter as the name and the increments the counter. If no parameter is given, 'this.name' is set to "???". Then the counter is set to 0. Next the function 'Student' is created. 'Student' takes two parameters. A new property called 'base' is created and set to the value of the 'Person' constructor. Then the 'base' method is called and the 'n' parameter is given as argument. Then 'this.base' is deleted. lastly 'this.studentid' is set to the second parameter. Dynamic inheritance is set up by executing "Student.prototype = new Person". This also increments the 'Person' counter. Then a new 'Student' object is created and assigned to the variable 'x'. Because 'Studen' inherits from 'Person', the 'Person' counter is incremented so when finaly 'x.count' is called, the value 2 is returned.

Object oriented hierarchy

The simple inheritance hierachy is done by using the built in prototype mechanism in JavaScript. To test this a div box with the outputs from the different calls has been created, as this creates the simplest test bench possible.

It is possible to ensure that Pet can never be instantiated through the one-shot closure:

```
// Global scope variables
// In actually they should probably be in a module
var Dog;
var Cat;
```

```
5
6   function() {
7     // This is a variable limited to the function scope.
8     var Pet = function() { /* ... */ };
9
10    Dog = function() { /* ... */ };
11    Cat = function() { /* ... */ };
12    Dog.prototype = new Pet()
13    /* ... other prototype actions ... */
14    Cat.prototype = new Pet()
15    /* ... other prototype actions ... */
16  }();
```

For concealing the name property the only option is to create a separate getName function for each subclass, thus eliminating the smart practicality with inheritance. This has been demonstrated in the Dog class, but would be quite impractical in the long run. The solution require the usage of "DontDelete" ($configurable : false$) and "Read-only" ($writable : false$).

The technique for making sound read-only is the same as for getName, albeit more practical as they aren't defined on the inheritance level.

Fun Sorting Game

Asynchronous loading

XMLHttpRequest

XMLHttpRequrest(xhr) provides some security features e.g: Same origin policy (sikkerhedspolitik). This tries to prevent cross domain attack and does not blindly parse code(You can still explicitly parse it). Thus it's still possible to implement the security errors. Furthermore using xhr also gives acess to http statuses which can be used to provide better feedback. A downside to xhr is that it can be rather tedious to code, but most libraries provide nice wrappers and if not using a library, it's fairly easy to code yourself.

iframe & Script Tags

Iframe and Script tags both have the advantage that they are rather light on code. On the other hand, neither iframe nor Script tags prevents cross origin attacks or execution of code. Usually this method is definitely not recomended but is in some cases used when browser permossions or similar prevents xhr. A common format in that case is JSONP.

Boss snooping

JSLint

The concept of validating JavaScript is justified because the interpretors operate by the "keep on trucking" philosophy. This in turn means that validation is limited at best. Thus JavaScript code is quite error prone and alot of pages have errors due to this fact.

The validation by JSLint is probably quite usefull, but seems to overly pedantic. For instance it require a whitespace in places where it has, and can never have, any semantic difference.